Proceedings of the 37th Italian Conference on Computational Logic (CILC 2022)

Roberta Calegari[1], Giovanni Ciatto[2] and Andrea Omicini[2]

[1]*Alma Mater Research Institute for Human Centered AI (AlmaAI), Alma Mater Studiorum—Università di Bologna*
[2]*Departement of Computer Science and Engineering (DISI), Alma Mater Studiorum—Università di Bologna*

The Italian conference on Computational Logic[1] (Convegno Italiano di Logica Computazionale, CILC) is the annual conference organised by the Italian group of researchers and users of logic programming[2] (Gruppo ricercatori e Utenti Logic Programming, GULP). Since the first event of the series, which was held in Genoa in 1986, the annual conference organised by GULP has been representing a major opportunity for researchers and practitioners working in the field of Computational Logic to meet and exchange ideas. The conference has broadened its horizons over the years, and today embraces topics that extend its reach beyond Computational Logic, such as declarative programming, knowledge representation, automated theorem proving, and virtually all applications of Computational Logic in the broader field of Artificial Intelligence.

The series of conferences organised by GULP has always been relevant to the GULP community, and Italian researchers and practitioners consider it as an indispensable tradition of the community. The 37th edition of the conference marks the attempt to return to normality after the COVID-19 pandemic and the consequent global health crisis. The conference was organised as a regular pre-pandemic event from June 29th, 2022 to July 1st, 2022 in Bologna, and it was attended in person by 45 participants. Other 15 participants joined the conference remotely by using a dedicated virtual meeting space, which was used to stream and record the presentations of accepted papers.

The community delivered a solid response to the urge for normality that characterised the days of the event. The 37th edition of the conference featured 29 presentations of high-quality papers, 22 of which were original works. Accepted contributions ranged from foundational and theoretical results to practical experiences, case studies, and applications, and they covered a wide range of relevant topics broadly related to Computational Logic. Accepted contributions included papers on agents and multi-agent systems, (constraint) logic programming, argumentation, and practical applications of logic programming. It is worth noting that accepted papers went through a strict evaluation process. Each original submission was evaluated by at least three anonymous reviewers from the Program Committee to ensure that the quality of papers

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
roberta.calegari@unibo.it (R. Calegari); giovanni.citatto@unibo.it (G. Ciatto); andrea.omicini@unibo.it (A. Omicini)
http://robertacalegari.apice.unibo.it (R. Calegari); https://about.me/gciatto (G. Ciatto)
0000-0003-3794-2942 (R. Calegari); 0000-0002-1841-8996 (G. Ciatto); 0000-0002-6655-3869 (A. Omicini)

[1]http://cilc2022.apice.unibo.it
[2]http://www.programmazionelogica.it

met the high-quality standards of the conference. Each non-original submission was reviewed by at least one anonymous reviewer from the Program Committee to check for coherence with the aims and scope of the conference. Note that, following the tradition of the conference, non-original submissions are not included in this proceedings.

The keynote speech was delivered by Manuel Hermenegildo, from the University Politecnica de Madrid, an international expert on Computational Logic. The keynote speech, titled "Prolog at 50", was a way for celebrating Prolog's 50th anniversary. After all these years, Prolog and Logic Programming are still relevant to higher-level programming and symbolic, explainable AI, with various implementations that keep on evolving, and new ones appearing. In this talk he provided an overview of the evolution of Prolog over these years since the original Marseille and Edinburgh versions, the current status of the language and its implementations, and some reflections on challenges and opportunities for the future. He also addressed some related issues such as the best ways of teaching Prolog and Logic Programming in general, applying Logic Programming technology to other languages, or the relation to other (logic and non-logic) programming languages.

The conference was enriched with a Technological Contest with the twofold purpose of *(i)* increasing the visibility of novel logic-based technologies with the logic programming community, and *(ii)* drawing a picture of the current state of the art of logic based technologies. There, 10 software contributions were presented.

Finally, a Special Session in honour of the former GULP President Gianfranco Rossi was successfully organised (mostly by Agostino Dovier), which brought back some good people and sweet memories from GULP past.

In conclusion, we would like to warmly thank all the people who contributed to CILC 2022. First, we would like to thank all the authors of the submitted papers, the invited speaker, the members of the Program Committee, and the anonymous reviewers. We are also grateful to the President of GULP, Stefania Costantini, and to all the members of the GULP Board, for their support and their fruitful suggestions. Finally, special thanks are due to all the attendees that joined the conference either in person or remotely for turning CILC 2022 into an occasion for lively discussions on relevant research topics and research challenges.

Roberta Calegari
Giovanni Ciatto
Andrea Omicini

Taking stock of available technologies for compliance checking on first-order knowledge

Livio Robaldo[1], Sotiris Batsakis[2], Roberta Calegari[3], Francesco Calimeri[4], Megumi Fujita[5], Guido Governatori[6], Maria Concetta Morelli[4], Giuseppe Pisano[3], Ken Satoh[5] and Ilias Tachmazidis[2]

[1]*Legal Innovation Lab Wales, Swansea University, Singleton Park, Sketty, Swansea SA2 8PP, United Kingdom*
[2]*Huddersfield University, Queensgate, Huddersfield HD1 3DH, United Kingdom*
[3]*University of Bologna, Via Zamboni, 33, 40126 Bologna, Italy*
[4]*University of Calabria, Via Pietro Bucci, 87036 Arcavacata, Rende, Italy*
[5]*National Institute of Informatics of Japan, 2 Chome-1-2 Hitotsubashi, Chiyoda City, Tokyo 101-8430, Japan*
[6]*Independent researcher*

Abstract
This paper analyses and compares some of the automated reasoners that have been used in recent research for compliance checking. We are interested here in formalizations at the *first-order* level. Past literature on normative reasoning mostly focuses on the *propositional* level. However, the propositional level is of little usefulness for concrete LegalTech applications, in which compliance checking must be enforced on (large) sets of individuals. This paper formalizes a selected use case in the considered reasoners and compares the implementations. The comparison will highlight that lot of further research still need to be done to integrate the benefits featured by the different reasoners into a single standardized first-order framework. All source codes are available at https://github.com/liviorobaldo/compliancecheckers

Keywords
Compliance checking, Normative reasoning, LegalTech

1. Introduction

LegalTech is experiencing growth in activity. Current LegalTech technologies mostly use Natural Language Processing (NLP) [1] or Machine Learning (ML) [2] to process documents and replicate legal decision-making.

However, ML is based on *statistical reasoning*: new cases are classified by similarity with the cases included in the training set. As a result, performance are intrinsically limited. Furthermore, and most important of all, as it is well-known ML tends to behave like a “black box” unable to explain its decisions and it can therefore lead to biases and other discriminatory outcomes: ML trained on biased datasets tend to replicate the same biases on new inputs.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
livio.robaldo@swansea.ac.uk (L. Robaldo); s.batsakis@hud.ac.uk (S. Batsakis); roberta.calegari@unibo.it (R. Calegari); francesco.calimeri@unical.it (F. Calimeri); kuma@nii.ac.jp (M. Fujita); kuma@nii.ac.jp (G. Governatori); kuma@nii.ac.jp (M. C. Morelli); g.pisano@unibo.it (G. Pisano); ksatoh@nii.ac.jp (K. Satoh); i.tachmazidis@hud.ac.uk (I. Tachmazidis)

In order to overcome the limits of ML, lot of recent research has been devoted to investigate approaches in symbolic AI. The idea is to plug into the ML-based system human-understandable symbols, i.e., concepts and other logical constructs, that enable forms of *logical reasoning* [3].

Logical formalization of norms requires *deontic operators* to represent the involved modalities (obligatory, permitted, prohibited) and *non-monotonic operators* fit to handle the central role of defeasibility in normative reasoning [4].

Formalizations found in past relevant literature are typically propositional, i.e. their basic components are whole propositions. However, propositions are of little usefulness for legal reasoning tasks needed within real-world LegalTech applications [5], due to their very limited expressivity. It is necessary to enhance the expressivity of the underlying logical format to the first-order level, fit to distinguish individuals from predicates and to allow the evaluation of deontic formulae to iterate over (large) sets of individuals.

This paper focuses on compliance checking with conflicting and compensatory norms. Compliance checking is the normative reasoning task of assessing whether a certain state of affairs complies or not with a set of norms. We are interested here in sets of norms where some of them conflict with others, for which it is necessary to establish preference criteria among them and to introduce defeasible operators to implement the overriding. On the other hand, compensatory norms are those that may be added "on the fly" to the set of norms in force whenever a violation occurs. For instance, if a traffic warden finds my car parked on the pavement, he will oblige me to pay a sanction. The payment of the sanction is then seen as a compensatory obligation for my illegal parking.

In this paper, we follow [6], which distinguishes between monotonic knowledge, encoded within an OWL ontology for the GDPR called PrOnto [7], and non-monotonic knowledge, i.e., the deontic and defeasible legal rules that implement the selected GDPR norms, encoded within a *separate* knowledge base in LegalRuleML [8]. Following [7], in this paper we will formalize the monotonic knowledge of our use case in OWL and we will define separate legal rules in the formats that we will compare.

2. The use case

In this paper, we use the following use case:

(1)
- **Art. 1**. The licensor grants the licensee a licence to evaluate the product.
- **Art. 2**. The licensee must not publish the results of the evaluation of the product without the approval of the licensor. If the licensee publishes these results without the approval, the material must be removed.
- **Art. 3**. The licensee must not publish comments about the evaluation, unless the licensee is permitted to publish the results of the evaluation.
- **Art. 4**. If the licensee is commissioned to perform an independent evaluation of the product, then the licensee is obliged to publish its results.

The use case in (1) is a simplification of use case 2 from [9]. We simplified the use case by removing all temporal information [10]. For instance, in the original version of Article 2 the licensee is obliged to remove the material *within 24 hours* after he had published it. We believe that adding time management will not constitute a relevant additional element of comparison; although we consider it as part of our future work, it is not in the scope of the present one. Thus, we interpret norms with respect to the state of affairs holding at the time "now". If "now" the licensee has published the material without the approval *and* he has "now" removed it, then he is "now" complying with Article 2.

According to standard legal theory [11], norms are formalized as if-then rules having a deontic statement (i.e., obligation, permission, or prohibition) in the consequent and, in the antecedent, the conditions for this statement to hold true.

Norms and corresponding if-then rules may be defeasible, in the sense that some of them may *override* others. Therefore, in order to properly formalize the articles in (1), we must also identify and formalize which norms override which other ones. In Art.1, the licensee is by default prohibited to evaluate the product; however, if he has the licence he is permitted to do so and this overrides the prohibition. Similarly, in Art.2, he is prohibited to publish the results unless he has the approval. In Art.3, the licensee is by default prohibited to publish comments unless he is permitted to publish the results. Finally, Art.4 states that in case the evaluation has been commissioned, the licensee is obliged to publish the results and this overrides any prohibition to do so.

On the other hand, as said above, some of the rules may *compensate* violation of others. These rules specify obligations that, when fulfilled, repair the non-compliance of other rules. The use case in (1) contains a single compensation in Article 2: if the licensee publishes the result of the evaluation without the approval, a new obligation holds for him: the licensee is obliged to remove them. In case this obligation is fulfilled, the violation has been repaired/compensated.

3. Formalizing norms at the first-order level

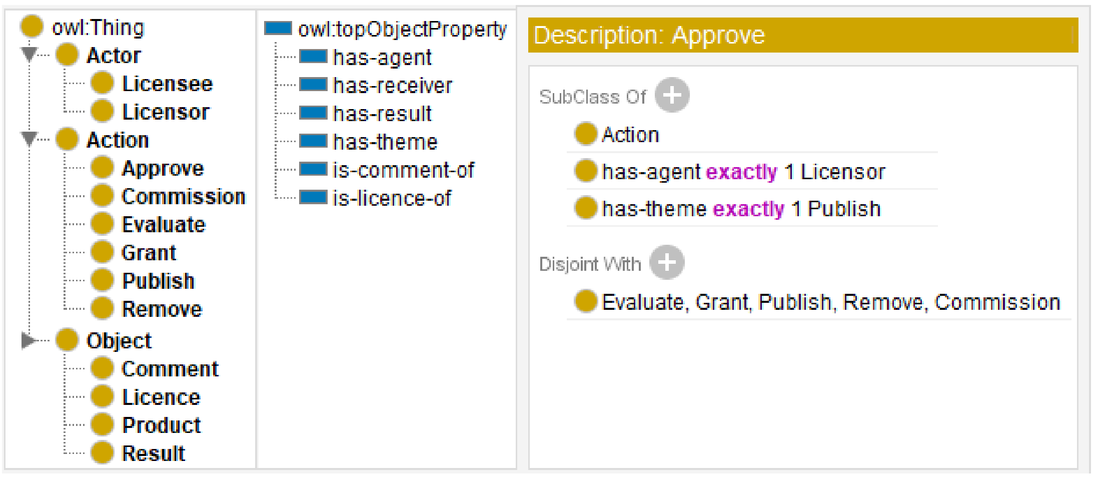
In this research work, we implemented the norms in (1) in six available formats for legal reasoning: SHACL [12], ASP-Core-2 [13], PROLEG [14], DLV [15], Arg2P [16], and SPINdle [17]. The if-then rules in (1) has been implemented at the first-order level, except in SPINdle in which the rules must be grounded.

Space constraints forbid us to report all formalizations in the paper. Thus, we will focus only on the two reasoners that have been identified as the "extremes" of the current state of the art: ASP-Core-2 and Arg2P. The reader is however invited to examine and execute all formalizations available on GitHub[1].

In line with [7], we stored all the monotonic knowledge of the use case within an OWL ontology, shown in Figure 1. The if-then rules representing the norms are separately formalized in the considered formats. These rules will all involve predicates corresponding 1:1 to the OWL resources in the ontology.

Nevertheless, the concepts in Figure 1 are not enough to formalize the norms. We also need concepts to model the deontic modalities, the defeasible rules, and the compensations. These concepts are inserted in a new separate ontology shown in Figure 2. To facilitate comprehension of the formulae, in Figure 2 we introduce subclasses of `Exception` whose names directly refer to the articles in the use case denoting the exceptions. The ontology includes a further object

[1]https://github.com/liviorobaldo/compliancecheckers


```
owl:Thing
  Actor
    Licensee
    Licensor
  Action
    Approve
    Commission
    Evaluate
    Grant
    Publish
    Remove
  Object
    Comment
    Licence
    Product
    Result
```

```
owl:topObjectProperty
  has-agent
  has-receiver
  has-result
  has-theme
  is-comment-of
  is-licence-of
```

Description: Approve

SubClass Of
- Action
- has-agent exactly 1 Licensor
- has-theme exactly 1 Publish

Disjoint With
- Evaluate, Grant, Publish, Remove, Commission

Figure 1: Classes and object properties of the reference ontology (implemented in Protégé); the class `Approve` is shown in full detail.

property `compensate`, not shown in Figure 2, that relates individuals of the class `Obligation` with individuals of the (union) class `Obligatory ∪ Prohibited`. Finally, we insert a further class `Violation` whose individuals will refer to the violated (and not compensated) obligations or prohibitions.

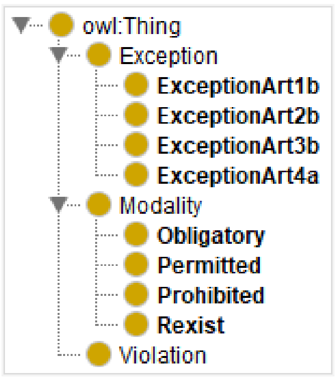

```
owl:Thing
  Exception
    ExceptionArt1b
    ExceptionArt2b
    ExceptionArt3b
    ExceptionArt4a
  Modality
    Obligatory
    Permitted
    Prohibited
    Rexist
  Violation
```

Figure 2: Extra classes to implement deontic modalities and defeasibility

3.1. Implementing the use case in ASP

In this subsection, we formalize the if-then rules as Answer Set Programming (ASP) rules. ASP is a widely used formalism for knowledge representation and reasoning; see [18] for an introduction. ASP is one of the most popular formalisms for AI, even at the industrial level [19]. Over the decades research has led to the definition of a variety of ASP "dialects", supported by corresponding ASP reasoners. The scientific community recently agreed on the definition of a standard input language for ASP systems, namely ASP-Core-2 [13].

ASP is a purely declarative formalism based on (if-then) rules. A given computational problem is solved in ASP by building a declarative logic program whose intended models, called *answer sets*, correspond 1:1 to the solution of the problem at hand. Since ASP is purely declarative, the order of the rules is irrelevant. Knowledge is just additive, and the ASP reasoner solves a program by searching for answer sets that satisfy all rules *at once*.

The ASP rule encoding Art.1 in (1), which states that the licensee is prohibited to evaluate the Product unless `exceptionArt1b` holds, is shown in (2)[2].

(2)
```
prohibited(Ev):-evaluate(Ev), hasAgent(Ev,X), licensee(X),
                  hasTheme(Ev,P), product(P), not exceptionArt1b(Ev).
```

"`not`" implements negation-as-failure. Thus, "`not exceptionArt1b(Ev)`" is true when the literal `exceptionArt1b(Ev)` is either false or unknown. `exceptionArt1b(Ev)` holds if the agent of `Ev` is granted a licence to evaluate the product. In such a case, the evaluation is permitted. However, the basic ASP language does not support conjunction of literals in rule heads; hence, in order to model such situations the typical approach consists of introducing a specific rule to define the condition, and then using it as antecedent of more than one rule:

(3)
```
condition1(Ev):-evaluate(Ev), hasAgent(Ev,X), licensee(X),
                  hasTheme(Ev,P), product(P), isLicenceOf(L,P),
                  licence(L), grant(Eg), rexist(Eg), hasTheme(Eg,L),
                  hasAgent(Eg,Y), licensor(Y), hasReceiver(Eg,X).

exceptionArt1b(Ev) :-condition1(Ev).

permitted(Ev) :-condition1(Ev).
```

Since `condition1(Ev)` is only used in these three if-then rules, indeed the first if-then rule in (3) is logically equivalent to a bi-implication, i.e., a definition.

Article 2 of the use case specifies both a prohibition and a compensatory obligation. Licensees are prohibited to publish the result of an evaluation unless this was approved by the licensor (first exception) or unless they were commissioned to perform an independent evaluation (second exception). Licensees who violate this prohibition are obliged to remove the published material.

The following rules define the prohibition described in Article 2a. The two mentioned exceptions are represented by the predicates `exceptionArt2b` and `exceptionArt4a`. We omit the ASP rules that entail `exceptionArt2b` as they are similar to (3). The ones that entail `exceptionArt4a` is shown below in (8).

(4)
```
condition2(Ep, X, R):-publish(Ep), hasAgent(Ep, X), licensee(X),
          hasTheme(Ep, R), result(R), hasResult(Ev, R), rexist(Ev),
          evaluate(Ev), not exceptionArt2b(Ep), not exceptionArt4a(Ep).

prohibited(Ep):-condition2(Ep, X, R).
```

[2]To model our use case, we will only consider `Action`(s) that exhaustively specify all (and only) their thematic roles. If these are unknown, the formula in (12) should not include thematic roles in the pre-conditions.

In order to model the compensatory obligation in Article 2c, we introduce a set of ASP rules that allow us to derive the same knowledge expressed by the following first-order logic well-formed formula:

(5) $\forall \mathtt{Ep} \forall \mathtt{X} \forall \mathtt{R}[(\mathtt{rexist(Ep)} \wedge \mathtt{condition2(Ep,X,R)}) \rightarrow \exists \mathtt{Y}[\mathtt{obligatory(Y)} \wedge \mathtt{remove(Y)} \wedge \mathtt{hasAgent(Y,X)} \wedge \mathtt{hasTheme(Y,R)} \wedge \mathtt{compensate(Y, Ep)}]]$

The ASP vocabulary does not include existential quantifiers. Thus, we make use of function symbols to simulate existential quantification via Skolemization. In particular, in this case, we use the function symbol "ca" (as for "compensatory action") and replace `Y` by `ca(Ep,X,R)`:

(6)
```
obligatory(ca(Ep,X,R)) :- rexist(Ep), condition2(Ep,X,R).
remove(ca(Ep,X,R)) :- rexist(Ep), condition2(Ep,X,R).
hasAgent(ca(Ep,X,R),X) :- rexist(Ep), condition2(Ep,X,R).
hasTheme(ca(Ep,X,R),R) :- rexist(Ep), condition2(Ep,X,R).
compensate(ca(Ep,X,R),Ep) :- rexist(Ep), condition2(Ep,X,R).
```

Article 3 defines the prohibition to publish comments on the evaluation of the product unless the licensee is allowed to publish the results of the evaluation of the product. The rules encoding Article 3 are shown in (7).

(7)
```
prohibited(Ep) :- publish(Ep), hasAgent(Ep, X), licensee(X),
                  hasTheme(Ep, C), comment(C), isCommentOf(C, Ev),
                  evaluate(Ev), rexist(Ev), not exceptionArt3b(Ep).

condition4(Ep) :- publish(Ep), hasAgent(Ep, X), licensee(X),
        hasTheme(Ep, C), comment(C), isCommentOf(C, Ev), rexist(Ev),
        evaluate(Ev), hasResult(Ev, R), hasTheme(Epr, R),
        hasAgent(Epr, X), publish(Epr), permitted(Epr).

exceptionArt3b(Ep) :- condition4(Ep).

permitted(Ep) :- condition4(Ep).
```

Finally, Article 4 establishes the obligation to publish the results of the evaluation in case this was commissioned, and thus an exception to Article 2.

(8)
```
condition5(Ep) :- publish(Ep), hasAgent(Ep, X), licensee(X),
                  hasTheme(Ep, R), result(R), hasResult(Ev, R),
                  evaluate(Ev), rexist(Ev), hasTheme(Ec, Ev),
                  commission(Ec), rexist(Ec).

exceptionArt4a(Ep) :- condition5(Ep).

obligatory(Ep) :- condition5(Ep).
```

3.1.1. Compliance checking via ASP rules.

The ASP rules shown in the previous subsection infer which actions are either prohibited or obligatory. Further ASP rules are then needed to infer the violations occurring in the state of affairs.

We remind that a violation occurs either in case an action is performed even if prohibited or in case an action is not performed even if obligatory. However, in both cases if the action is associated with a compensatory obligation and the latter was performed, the former does not indeed trigger any violation. The ASP rules in (9) are able to carry out the desired inferences.

(9)
```
compensated(X) :- compensate(Y, X), rexist(Y).
violation(viol(X)) :- obligatory(X), not rexist(X), not compensated(X).
violation(viol(X)) :- prohibited(X), rexist(X), not compensated(X).
referTo(viol(X), X) :- violation(viol(X)).
```

Finally, we add the ASP rule in (10) in order to intercept the occurrence in the state of affairs of a removal action that has the properties required by the removal action denoted by `ca(Ep,X,R)` in (6). The rule in (10) is needed to "solve" the existential quantification, represented as a Skolemized functional symbol.

(10)
```
rexist(ca(Ep,X,R)) :- remove(ca(Ep,X,R)),
                hasTheme(ca(Ep,X,R),R), hasAgent(ca(Ep,X,R),X),
                rexist(Er), remove(Er), hasTheme(Er,R), hasAgent(Er,X).
```

In other words, the rule in (10) "solves" the existential quantification (represented here as a functional term) by searching for an action with the same type and the same thematic roles and that really exists in the model. If this action is found, also the functional term `ca(Ep,X,R)` is asserted as really existing.

3.2. Implementing the use case in Arg2P

Several modern approaches to legal reasoning are based on structured argumentation [20]. These approaches provide an extra layer to the representation of the inferences by including therein the graph of the arguments that either support or reject the conclusions. Although argumentation offers more functionalities than what we need to model our use case, we still decided to consider it in our analysis given the prominent role that it is increasingly assuming in LegalTech.

In this paper we consider Arg2P [21], a lightweight Prolog-based implementation for structured argumentation in compliance with the micro-intelligence definition [16]. The research in Arg2P aims to identifying different functionalities offered by available defeasible reasoners and to allow the users to configure Arg2P on the ones they need in their domain and for the purposes of their projects.

Arg2P format allows to encode labelled defeasible inference rules each from a conjunction of premises to a conclusion. Overriding among rules is achieved via superiority relations. Arg2P format also includes modal operators[3], “o” and “p”, respectively stating whether an action is obligatory or permitted.

Article 1 of the use case is formalized via the following Arg2P formulae, which corresponds to the ASP formulae in (2) and (3).

```
(11)  art1a: evaluate(Ev), hasAgent(Ev,X), licensee(X),
             hasTheme(Ev,P), product(P) => o(-evaluate(Ev)).
      art1b: evaluate(Ev), hasAgent(Ev,X), licensee(X), licence(L),
             hasTheme(Ev,P), product(P), isLicenceOf(L,P),
             hasTheme(Eg,L), hasAgent(Eg,Y), licensor(Y), grant(Eg),
             rexist(Eg), hasReceiver(Eg,X) => p(evaluate(Ev)).
      sup(art1b, art1a).
```

If the licensor grants a licence, the rules in (11) derive that the evaluation is both prohibited (`o(-evaluate(Ev))`) and permitted (`p(evaluate(Ev))`); however, as the superiority relation `sup(art1b, art1a)` states that the rule with label `art1b` is stronger than the rule with label `art1a`, only `p(evaluate(Ev))` is inferred.

The if-then rule that encodes the prohibition in Article 2 is shown in (12).

```
(12)  art2aPart1: evaluate(Ev), rexist(Ev), hasResult(Ev,R),
                  result(R), publish(Ep), hasAgent(Ep,X),
                  licensee(X), hasTheme(Ep,R) => condition2(Ep,X,R).
      art2aPart2: condition2(Ep,X,R) => o(-publish(Ep)).
```

The rule implementing the obligations from Article 4 is then:

```
(13)  art4a: publish(Ep), hasAgent(Ep,X), licensee(X), result(R),
             hasTheme(Ep,R), hasResult(Ev,R), evaluate(Ev), rexist(Ev),
             hasTheme(Ec,Ev), commission(Ec), rexist(Ec) => o(publish(Ep)).
      sup(art4a, art2aPart2).
```

The superiority relation in (13) blocks the first rule in (12) in case the evaluation of the product has been commissioned. A similar rule, which we omit in this paper, blocks the first rule in (12) in case the licensor approved the publishing.

In order to represent the rest of Article 2, we introduce a set of rules that parallel the ASP ones in (6) above. These are shown in (14).

```
(14)  art2cPart1: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
                  => o(remove(ca(Ep,X,R))).
      art2cPart2: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
                  => remove(ca(Ep,X,R)).
```

[3]See https://pika-lab.gitlab.io/argumentation/arg2p-kt/wiki/syntax

```
art2cPart3: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
              => hasTheme(ca(Ep,X,R),R).
art2cPart4: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
              => hasAgent(ca(Ep,X,R),X).
art2e: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
         => compensate(ca(Ep,X,R),Ep).
```

Finally, Article 3 of the use case is formalized as in (15): the publishing of the comments is prohibited unless the publishing of the results is permitted.

```
(15)  art3a: publish(Ep), hasAgent(Ep,X), licensee(X),
             hasTheme(Ep,C), comment(C), isCommentOf(C,Ev),
             evaluate(Ev), rexist(Ev) => o(-publish(Ep)).
      art3b: publish(Ep), hasAgent(Ep,X), licensee(X), comment(C),
             hasTheme(Ep,C), isCommentOf(C,Ev), hasResult(Ev,R),
             evaluate(Ev), rexist(Ev), hasTheme(Epr,R), hasAgent(Epr,X),
             publish(Epr), p(publish(Epr)) => p(publish(Ep)).
      sup(art3b, art3a).
```

3.2.1. Compliance checking via Arg2P rules.

Arg2P represents obligatory, prohibited, and permitted actions via two modal operators "o" and "p". Since Arg2P's input format does not allow to quantify over the predicates outscoped by "o", we must assert a different rule for *each* action that may be prohibited. In our use case, two actions may be prohibited: the evaluation of the product and the publishing of either its results or comments about it. Each of these two actions is associated with a different Arg2P rule that detects the violation of its prohibition. "~" is the Arg2P operator for negation-as-failure.

```
(16)   ccRuleEv: o(-evaluate(Ev)), rexist(Ev), ~(compensated(Ev))
                   => violation(viol(Ev)).
       ccRuleEp1: o(-publish(Ep)), rexist(Ep), ~(compensated(Ep))
                    => violation(viol(Ep)).
```

On the other hand, in our use case there are two actions that may be obligatory: the publishing of the results, which is obligatory in case the evaluation has been commissioned, and the removal of the results, which is obligatory in case the licensee publishes the results even if he was not allowed to do so.

```
(17)  ccRuleEp2: o(publish(Ep)), ~(rexist(Ep)), ~(compensated(Ep))
                   => violation(viol(Ep)).
      ccRuleEr: o(remove(Er)), ~(rexist(Er)), ~(compensated(Er))
                  => violation(viol(Er)).
```

Finally, we need Arg2P rules to infer when the remove action `ca(Ep,X,R)` really exists and, consequently, when the prohibited publishing have been compensated:

```
(18)  ccRuleComp1: remove(ca(Ep,X,R)), hasTheme(ca(Ep,X,R),R),
              hasAgent(ca(Ep,X,R),X), rexist(Er), remove(Er),
              hasTheme(Er,R), hasAgent(Er,X) => rexist(ca(Ep,X,R)).
      ccRuleComp2: compensate(Y,X), rexist(Y) => compensated(X).
```

4. Comparison, discussion, and future works

We developed a dataset generator that creates synthetic ABox(es) in the input format of each reasoner. The reasoners are then executed on these ABox(es) to compare their performance. The GitHub repository contains instructions to recreate the datasets locally and to execute the reasoners on them.

Table 1 shows the time performance on three datasets respectively including 10, 30, and 50 states of affairs. All experiments reported in this paper were run on a PC with Intel(R) Core(TM) at 1.8 GHz, 16 GB RAM, and Windows 10.

Table 1
Time performance of the compliance checkers

Size	SHACL	ASP (clingo)	ASP (DLV2)	DLV	PROLEG	Arg2P	SPINdle
10	0.091s	0.019s	0.0552s	0.0347s	0.398s	398.338s	0.063s
30	0.122s	0.025s	0.0337s	0.0505s	0.631s	1039.668s	0.099s
50	0.148s	0.051s	0.0553s	0.097s	1.374s	1927.389s	0.187s

From the results reported in Table 1, it is evident that PROLEG and, in particular, Arg2P are much slower than the other reasoners. We were indeed surprised ourselves that Arg2P's time performance were *so much* lower.

Since Arg2P is one of the most modern implemented reasoners for structured argumentation, the assessed slow performance definitely demands for much further research. Structured argumentation has been mainly studied so far from a theoretical point of view but it is time now to research ways of making the theoretical findings usable in practice. This could be perhaps achieved by modeling problems in argumentation precisely as problems in ASP, in order to make the most of the format's efficiency, a solution already advocated in [18].

Similar considerations hold for PROLEG. However, contrary to Arg2P, PROLEG is not a stand-alone legal reasoner. It is a library that must be loaded within other Prolog reasoners, e.g., SWI Prolog[4]. Thus, carrying out further research to improve PROLEG efficiency most likely amounts to carrying out further research to improve the efficiency of reasoners for standard Prolog.

[4]https://www.swi-prolog.org

On the other hand, although computational performance is of primary importance in the big data era, it cannot be the sole criterion for comparison.

Before using the formulae, these must be built and checked/debugged. The use case in (1) is just a constructed example inspired by existing norms. Still, it required us considerable time to be formalized. Therefore, other parameters such as human-readability, easy of editing, explainability, etc. must be considered.

Unfortunately, although ASP is so efficient, achieving explainability in ASP could be difficult because, as explained in subsection 3.1 above, ASP is a declarative language in which the reasoner tries to satisfy all rules *at once*. The returned answer set does not specify which rules have been applied to obtain the facts within the answer set. This knowledge must be inferred through an additional "reverse engineering" process, from the returned answer set to the asserted rules.

Achieving explainability in ASP is a matter of ongoing research [22]. Several techniques and methodologies to debug answer-set programs have been proposed, among which [23] and [24]. The common insight of these solutions is to add an extra-layer that relates the facts in the returned answer set with the rules that derive them, thus allowing to trace the inferential process.

Furthermore, ASP uses negation-as-failure in place of the superiority relations used in Arg2P and PROLEG. The latter have been proved to be more readable and intuitive than the former: while superiority relations straightforwardly allow to encode the directed acyclic graph representing which rules override which other ones, negation-as-failure requires to introduce additional special predicates that explicitly refer to the exceptions. As these additional predicates increase in number along with the number of exceptions, it might be harder for a human to keep track and organize them when translating large sets of norms.

On the other hand, while we were formalizing the use case in the different formats we realized that modal operators, such as the operators "o" and "p" of Arg2P, are rather difficult to use in conjunction with first-order formulae. On the contrary, unary first-order predicates applied to terms that directly refer to the actions appear to be easier for editing, reading, and debugging the formulae.

Arg2P's modal operators can outscope a *single* predicate. On the other hand, for representing the actions together with their thematic roles we need a *conjunction* of predicates, e.g., `publish(Ep)`, `hasAgent(Ep,X)`, `licensee(X)`, etc.

In our view, the only way to achieve the desired truth conditions in the current version of Arg2P is then to assert the conjunction of predicates in the antecedent of the rule and the modal operator applied to the "main" predicate in the consequent. This was done, for instance, for rule "`art4a`" in (13).

Moreover, we observe that allowing the Arg2P operator "o" to also accept a conjunction of predicates does not appear to solve the problem so simply. In Standard Deontic Logic (SDL)[5], which inspired the definition of these operators, the axiom $\mathrm{o}(P_1, P_2) \rightarrow (\mathrm{o}(P_1), \mathrm{o}(P_2))$ holds. Thus, we may derive, for instance:

```
(19)   o(publish(x,r), licensee(x), result(r))
                           => o(publish(x,r)), o(licensee(x)), o(result(r))
```

[5]https://plato.stanford.edu/entries/logic-deontic/#StanDeonLogi

(19) means that it is obligatory for the individual x to be a licensee and for the individual r to be a result, which sounds weird and counter-intuitive.

Also the modal operators of the LegalRuleML standard[6] suffer from the same problem. Possible solutions within future versions of the standard are still under discussion within the LegalRuleML technical committee (TC)[7].

These counter-intuitive derivations are not found with propositional symbols, e.g., in SPINdle. It is a problem related to the use of modal deontic operators in conjunction with *first-order* formulae, for which one should perhaps define an alternative semantics for the modal operators that does not encompass the SDL axiom in (19). However, this solution requires much further research to properly investigate whether it could lead or not to other counter-intuitive effects.

5. LegalRuleML

Artificial Intelligence is currently in a transition phase, from standard solutions based on Machine Learning to novel solutions based on symbolic reasoning fit to support human centricity, i.e., explainability and human-readability.

This is true also in AI for the legal domain, as witnessed by lot of recent literature, e.g., [25] and [26], as well as related initiatives such as the yearly EXplainable & Responsible AI in Law (XAILA) workshop[8].

Building symbolic knowledge is highly time-consuming, especially in LegalTech where the knowledge originates from norms written in natural language. Moreover, norms from real legislation are highly dependent on the legal domain they regulate (finance, health, etc.); thus, their proper formalization must necessarily involve lawyers or other domain experts, many of whom are indeed unfamiliar with logic and technical details.

In our view, the involvement of domain experts towards the creation of large knowledge bases of machine-readable formulae associated with existing legislation might be achieved only via a *standardized methodology*, from the norms in natural language to the executable formalizations in some legal reasoner.

In the future, we intend to define such a methodology around LegalRuleML, which became an OASIS standard very recently, i.e., on August 30th, 2021[9].

LegalRuleML is an XML-based semi-formal language aiming at enhancing the interplay between experts in law and experts in logic. By "semi-formal" we intend that no formal model-theoretic semantics is associated with LegalRuleML. Well-formed LegalRuleML representations need to be translated into another language having such a semantics, e.g., the input formats of the legal reasoners considered above, similarly to what is done in Reaction RuleML 1.0 via the so-called "semantics profiles" [27].

Still, LegalRuleML defines a specification, in terms of an XML vocabulary and composition rules, that is able to represent the particularities of the legal normative rules with a rich,

[6]https://docs.oasis-open.org/legalruleml/legalruleml-core-spec/v1.0/os/legalruleml-core-spec-v1.0-os.html#_Toc38017882

[7]Personal communications between Livio Robaldo, Guido Governatori, Monica Palmirani, and Adam Wyner, during the recent activities of the LegalRuleML TC.

[8]https://www.geist.re/xaila:start

[9]See https://www.oasis-open.org/standard/legalruleml

articulated, and meaningful markup language.

The advocated annotations in LegalRuleML may be facilitated via a special graphical editor that allows to compose the if-then rules and store them in the XML standard. Something similar has been recently done in [28], which present a graphical editor to annotate norms in reified I/O logic [29], a novel deontic logic based on reification [30] [31] which features a computational complexity lower than standard approaches based on possible-world semantics [32]. The LegalRuleML annotations so produced can be then automatically translated in a computational language to check the compliance of the denoted norms with respect to a given state of affairs.

In our future works, we will implement advanced editors for LegalRuleML, as well as translation algorithms from LegalRuleML to executable formats. Further modules may be developed as well, e.g., NLP procedures to suggest draft LegalRuleML representations that the annotators must validate or amend.

These editors will allow us to promote annotation campaigns involving domain experts or even law students. These campaigns will be possibly part of the future activities of the LegalRuleML technical committee and they are expected to stimulate and guide future research towards standardized and interoperable solutions for automated legal reasoning.

6. Conclusions

In this paper, we investigated some of current technologies for compliance checking at the *first-order* level, with conflicting and compensatory norms.

Most implemented legal reasoners, first of all SPINdle, are propositional. Nevertheless, propositional reasoning is too limited for existing applications. Thus, we investigated and compared some of main current reasoning languages with respect to a shared use case and a shared vocabulary of atomic predicates each of which corresponds to a concept in the ontology from Figure 1.

So far these reasoning languages were mostly studied in isolation. Investigating them altogether with respect to a shared use case and a shared vocabulary of predicates allowed to highlight their respective peculiarities.

Arg2P and PROLEG are inefficient, in particular Arg2P. However, these two reasoners are currently the only ones able to explain their derivations. Conversely, ASP is very efficient but lacks explainability because the declarative nature of the language itself makes it difficult to debug the inferences.

Finally, we observed that some of the operators used in the different formats, e.g., modal operators and negation-as-failure, could be hard to manipulate. Readability may be improved by defining one-to-one translation procedures from/to LegalRuleML. LegalRuleML aims at being an easy and intuitive formal language to allow domain experts, even those unfamiliar with logic and technical details, to contribute in the construction of large knowledge bases of legal rules.

Acknowledgments

Livio Robaldo has been supported by the Legal Innovation Lab Wales operation within Swansea University's Hillary Rodham Clinton School of Law. The operation has been part-funded by the European Regional Development Fund through the Welsh Government.

References

[1] R. Nanda, L. Di Caro, G. Boella, H. Konstantinov, T. Tyankov, D. Traykov, H. Hristov, F. Costamagna, L. Humphreys, L. Robaldo, M. Romano, A unifying similarity measure for automated identification of national implementations of european union directives, in: Proc. of the 16th Edition of the International Conference on Articial Intelligence and Law (ICAL 2017), Association for Computing Machinery, 2017.

[2] G. Boella, L. Di Caro, D. Rispoli, L. Robaldo, A system for classifying multi-label text into eurovoc, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law, ICAIL '13, ACM, 2013, pp. 239–240.

[3] C. Bartolini, A. Giurgiu, G. Lenzini, L. Robaldo, Towards legal compliance by correlating standards and laws with a semi-automated methodology, in: BNCAI, volume 765 of *Communications in Computer and Information Science*, Springer, 2016.

[4] D. Gabbay, J. Horty, X. Parent, R. van der Meyden, L. van der Torre, Handbook of Deontic Logic and Normative Systems, College Publications, 2013.

[5] G. Antoniou, G. Baryannis, S. Batsakis, G. Governatori, M. B. Islam, Q. Liu, L. Robaldo, G. Siragusa, I. Tachmazidis, Large-scale legal reasoning with rules and databases, Journal of Applied Logics - IfCoLog Journal 8 (2021) 911–940.

[6] M. Palmirani, G. Governatori, Modelling legal knowledge for GDPR compliance checking, in: Legal Knowledge and Information Systems (JURIX), 2018.

[7] M. Palmirani, M. Martoni, A. Rossi, C. Bartolini, L. Robaldo, Pronto: Privacy ontology for legal compliance, in: EU conference on digital government, 2018.

[8] T. Athan, G. Governatori, M. Palmirani, A. Paschke, A. Wyner, LegalRuleML: Design Principles and Foundations, Springer International Publishing, 2015.

[9] S. Batsakis, G. Baryannis, G. Governatori, I. Tachmazidis, G. Antoniou, Legal representation and reasoning in practice: A critical comparison, in: Legal Knowledge and Information Systems, JURIX, 2018.

[10] L. Robaldo, T. Caselli, I. Russo, M. Grella, From Italian text to TimeML document via dependency parsing, in: Proc. of Computational Linguistics and Intelligent Text Processing (CICLing 2011)., 2011.

[11] G. Sartor, Legal concepts as inferential nodes and ontological categories, Artificial Intelligence and Law 17 (2009) 217–251.

[12] L. Robaldo, Towards compliance checking in reified I/O logic via SHACL, in: J. Maranhão, A. Z. Wyner (Eds.), Proc. of 18th International Conference for Artificial Intelligence and Law (ICAIL 2021), ACM, 2021.

[13] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, Asp-core-2 input language format, Theory and Practice

of Logic Programming 20 (2020) 294–309.
[14] K. Satoh, K. Asai, T. Kogawa, M. Kubota, M. Nakamura, Y. Nishigai, K. Shirakawa, C. Takano, PROLEG: An Implementation of the Presupposed Ultimate Fact Theory of Japanese Civil Code by PROLOG Technology, in: T. Onada, D. Bekki, E. McCready (Eds.), New Frontiers in Artificial Intelligence, 2011.
[15] F. Buccafurri, W. Faber, N. Leone, Disjunctive logic programs with inheritance, Theory and Practice of Logic Programming 2 (2002).
[16] R. Calegari, A. Omicini, G. Pisano, G. Sartor, Arg2P: an argumentation framework for explainable intelligent systems, J. of Logic and Computation 32 (2022).
[17] H.-P. Lam, G. Governatori, The Making of SPINdle, in: Proc. of International Symposium on Rule Interchange and Applications (RuleML), 2009.
[18] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103.
[19] K. Reale, F. Calimeri, N. Leone, F. Ricca, Smart devices and large scale reasoning via ASP: tools and applications, in: J. Cheney, S. Perri (Eds.), Practical Aspects of Declarative Languages - 24th International Symposium, 2022.
[20] H. Prakken, G. Sartor, Law and logic: A review from an argumentation perspective, Artificial Intelligence and Law 227 (2015) 214–245.
[21] M. Billi, R. Calegari, G. Contissa, F. Lagioia, G. Pisano, G. Sartor, G. Sartor, Argumentation and defeasible reasoning in the law, J 4 (2021).
[22] J. Dauphin, K. Satoh, Explainable ASP, in: M. Baldoni, M. Dastani, B. Liao, Y. Sakurai, R. Zalila-Wenkstern (Eds.), Proc. of 22nd international conference on Principles and Practice of Multi-Agent Systems (PRIMA 2019), volume 11873 of *Lecture Notes in Computer Science*, Springer, 2019.
[23] J. Oetsch, J. Pührer, H. Tompits, Stepwise debugging of answer-set programs, Theory and Practice of Logic Programming 18 (2018).
[24] B. Cuteri, C. Dodaro, F. Ricca, Debugging of answer set programs using paracoherent reasoning, in: A. Casagrande, E. G. Omodeo (Eds.), Proc. of the 34th Italian Conference on Computational Logic (CILC 2019), volume 2396 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019.
[25] B. Waltl, R. Vogl, Explainable artificial intelligence – the new frontier in legal informatics, Jusletter IT 22 (2018).
[26] A. Bibal, M. Lognoul, A. de Streel, B. Frénay, Legal requirements on explainability in machine learning, Artificial Intelligence and Law 29 (2021) 149–169.
[27] A. Paschke, Reaction ruleml 1.0 for rules, events and actions in semantic complex event processing, in: A. Bikakis, P. Fodor, D. Roman (Eds.), Rules on the Web. From Theory to Applications, Springer International Publishing, 2014.
[28] L. Robaldo, C. Bartolini, M. Palmirani, A. Rossi, M. Martoni, G. Lenzini, Formalizing gdpr provisions in reified i/o logic: the dapreco knowledge base., The Journal of Logic, Language, and Information 29 (2020).
[29] L. Robaldo, X. Sun, Reified input/output logic: Combining input/output logic and reification to represent norms coming from existing legislation., The Journal of Logic and Computation 7 (2017).
[30] J. R. Hobbs, Deep lexical semantics, in: Proc. of the 9th International Conference on

Intelligent Text Processing and Computational Linguistics (CICLing-2008), Haifa, Israel, 2008.

[31] L. Robaldo, Distributivity, collectivity, and cumulativity in terms of (in)dependence and maximality., The Journal of Logic, Language, and Information 20(2) (2011) 233–271.

[32] X. Sun, L. Robaldo, On the complexity of input/output logic., The Journal of Applied Logic 25 (2017) 69–88.

GPU Parallelism for SAT Solving Heuristics[⋆]

Michele Collevati[1], Agostino Dovier[1,2] and Andrea Formisano[1,2,*]

[1]CLPLab - DMIF - Università di Udine, via delle Scienze 206, 33100 Udine, Italy
[2]GNCS-INdAM, piazzale Aldo Moro 5, 00185 Roma, Italy

Abstract

Modern SAT solvers employ a number of smart techniques and strategies to achieve maximum efficiency in solving the Boolean Satisfiability problem. Among all components of a solver, the branching heuristics plays a crucial role in affecting the performance of the entire solver. Traditionally, the main branching heuristics that have appeared in the literature have been classified as *look-back* heuristics or *look-ahead* heuristics. As SAT technology has evolved, the former have become more and more preferable, for their demand for less computational effort.

Graphics Processor Units (GPUs) are massively parallel devices that have spread enormously over the past few decades and offer great computing power at a relatively low cost. We describe how to exploit such computational power to efficiently implement look-ahead heuristics. Our aim is to "rehabilitate" these heuristics, by showing their effectiveness in the contest of a parallel SAT solver.

Keywords

SAT Solving, Branching Heuristics, GPU parallelism

1. Introduction

The central point of either DPLL [1] or CDCL [2] SAT solvers is the choice of the successive variable to be assigned (*variable selection heuristics*) and the choice of the Boolean value to be attempted first (*polarity selection heuristics*). The algorithms for implementing the two choices are called *branching heuristics*, and, apart from the naive ones (e.g., leftmost variable, random choice, etc.), they can be classified as *look-back* heuristics or *look-ahead* heuristics. The former are, in general, easier to implement, since it is sufficient to collect and maintain minimal information about the evolution of the computation. Look-ahead heuristics require a (partial) exploration of the "future" of the computation in order to determine the potential impact of alternative choices the solver can make at a choice point and this can be computationally expensive. This is a reason why look-ahead heuristics have largely been abandoned in modern solvers, in favor of the "lighter" (and, somehow, possibly coarser) look-back heuristics.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

[⋆] Research partially supported by Fondazione Friuli/Università di Udine project on *Artificial Intelligence for Human Robot Collaboration* and by projects INDAM GNCS-2020 *NoRMA* and INDAM GNCS-2022 InSANE (CUP_E55F22000270001).

[*] Corresponding author.

michele.collevati@protonmail.com (M. Collevati); agostino.dovier@uniud.it (A. Dovier); andrea.formisano@uniud.it (A. Formisano)
0000-0001-7958-7841 (M. Collevati); 0000-0003-2052-8593 (A. Dovier); 0000-0002-6755-9314 (A. Formisano)

GPU manufacturer NVIDIA, through its platform called *Computing Unified Device Architecture (CUDA)* [3], is a leading pioneer in GPU-computing. CUDA, unveiled in 2006, is a general-purpose parallel computing platform and programming model that leverages the parallel computing engine of NVIDIA GPUs. It can be programmed in C or C++ and it enables the development of applications that scale their parallelism transparently and take advantage of the growing number of CPU and GPU cores. Although initially GPU were used for graphical purposes, e.g., video games, they are nowadays widely used in Deep Learning computation. A stream of works trying to exploit them for SAT/ASP solving exists [4, 5, 6, 7].

In this paper, we describe how to exploit the computational power of GPUs by developing a CUDA C library that implements the look-ahead heuristics. In a sense, our aim is to "rehabilitate" these heuristics, by showing their effectiveness in the contest of a parallel SAT solver. In Section 2 we introduce the main notions and notation used in the paper. Section 3 describes the main ideas behind the GPU implementation of the look-ahead heuristics. In Section 4 we report on the experiments made using our implementation with a DPLL and a CDCL solver. Conclusions are drawn in Section 5.

2. Background

Let $\mathcal{V}$ be a (denumerable) set of variables. If $x \in \mathcal{V}$ then x and $\neg x$ are said *literals*. A disjunction $\omega = (\ell_1 \vee \cdots \vee \ell_k)$ of literals is said a *clause*. If $k = 1$ the clause ω is said *unit clause* (or, simply, *unit*). A Boolean formula Φ in *Conjunctive Normal Form (CNF)* is a conjunction $(\omega_1 \wedge \cdots \wedge \omega_h)$ of clauses. As common, for denotational convenience, we might refer to Φ as a set of clauses and, similarly, to a clause ω as a set of literals.

A *(partial) Boolean assignment* σ is a mapping from $X \subseteq \mathcal{V}$ to $\{\mathsf{false}, \mathsf{true}\}$. An assignment can be applied to literals, clauses, and formulas, and evaluated using the classical semantics of propositional connectives $\neg, \vee, \wedge$. In particular, for any clause ω, $\sigma(\omega) = \mathsf{true}$ if and only if $\sigma(\ell) = \mathsf{true}$ for some $\ell \in \omega$. In such case, we say that ω is *satisfied* by σ. If $\sigma(\ell) = \mathsf{false}$ for each $\ell \in \omega$, then $\sigma(\omega) = \mathsf{false}$. Given a set of clauses Φ (i.e., a CNF Boolean formula), an assignment σ is a *solution* for Φ if σ satisfies all $\omega \in \Phi$. The *Boolean Satisfiability (SAT)* problem is the problem of establishing whether a solution exists for a given Φ.

An assignment σ may be partial. Namely, it might be the case that $\sigma(\ell)$ is not defined for some $\ell \in \mathcal{V}$. In case some literals in a clause ω are not assigned by σ, and ω is not satisfied by σ, the clause is said *unresolved* (w.r.t. σ). An unresolved clause ω with only one undefined literal is said *unit* (w.r.t. σ).

2.1. SAT Solving

The relevance of SAT for the $\mathcal{P}$ *versus* $\mathcal{NP}$ problem is clear since the seminal papers by Cook and Levin [8, 9]. However, the research of an automatic procedure capable of solving concrete instances of SAT was already one of the most important research area of theorem proving. In this context, a milestone was posed by Davis and Putnam [10] developing a procedure later refined for reducing space occupation by their co-authors Logemann and Loveland [1]. The algorithm, known as *DPLL*, combines three stages:

1. *(unit) propagation*: deterministically infer values for variables under a given partial assignment σ: whenever a clause ω is unit w.r.t. σ, extend σ so as to satisfy ω;
2. *choice*: non-deterministically choose a not yet assigned variable x, assign it a value among false and true, and extend the current partial assignment accordingly;
3. *backtracking*: if a failure is reached because no solutions were found under the current assignment, backtrack the last choice made in assigning a variable.

The simplest possible solver starts from the empty assignment (all variables are unassigned) and alternates propagation and choice steps. At each moment in time, the number of active choices performed is the *decision level* currently reached in the search for a solution. The search proceeds until either a solution is found (i.e., an assignment satisfying all clauses) or a clause gets falsified by the current (partial) assignment. In this case the computation backtracks and the decision level is decreased (undoing the effects of the last choice). Hence, the solver proceeds by visiting a tree-shaped search space and the decision level is the depth currently reached in the search tree.

Implementing a DPLL solver requires the selection of the algorithms to perform step (2). Concretely, one has to choose two heuristics to be used in *choice points* to select

- the variable to be assigned, called *Variable Selection Heuristics (VSH)*, and
- the truth value to be attempted first, called *Polarity Selection Heuristics (PSH)*.

A second family of SAT solvers extends the idea of DPLL by analyzing the reasons why an assignment has lead the search to a failure. Solvers of this family are called *Conflict-Driven Clause-Learning (CDCL)* solvers. These solvers proceed as DPLL until a failure is detected. Then, a step called *conflict analysis* is performed to detect a reason for the failure, namely, a set of variable assignments (made by the choice steps (2)) that conjunctively prevent the satisfaction of some clauses of the input formula Φ. This set of variable assignments identifies a new clause that can be *learned* and added to Φ. The rational is that any learned clause ω is a logical consequence of Φ. Hence, if σ is a solution of Φ, it is also a solution of $\Phi \wedge \omega$. After a new clause is learned, the solver *backjumps* to a decision level preceding (at least some of) conflicting choices (undoing their effects on the current assignment). Each learned clause is expected to speed up the subsequent search because it prevents the solver from making the same failing set of assignments again. Here, usually, unit propagation enters into play: whenever all but one of the assignments of such set are done, the presence of the learned clause forces a different value selection for the remaining assignment. Clearly, "short" learned clauses are more effective in speeding up the search by reducing the search space. We refer the reader to [2] for a detailed formal description of CDCL solvers.

The branching heuristics can be partitioned into two families:

1. *look-back* heuristics, which rank variables on the basis of the computation performed till the choice point;
2. *look-ahead* heuristics, which rank variables on the basis of the effect of their assignment in the subsequent part of the computation.

Look-back heuristics are, in general, easier to implement. It suffices to gather, during the computation, some information about the assignments and their effects (e.g., the simplifications of the input formula enabled by the performed assignments, or some statistics about failures, etc.). The overhead involved by these heuristics is small w.r.t. the whole computation. Conversely, look-ahead heuristics require a (partial) exploration of the "future" of the computation in order to determine the potential impact of alternative choices the solver can make at a choice point. This usually amounts to speculatively performing some steps of unit propagation. Albeit, in principle, look-ahead heuristics may lead to better choices (those that speed up the search the most), they also involve higher computational overhead, especially in a sequential implementation. This is a reason why look-ahead heuristics have largely been abandoned in modern solvers, in favor of the "lighter", but possibly coarser, look-back heuristics.

2.2. Look-ahead Heuristics

Look-ahead heuristics score variables depending on the effect their assignment has on the *current* state of the search. These heuristics can be considered as greedy algorithms: they evaluate, with respect to some *estimation function*, the alternative possible choices and select the most effective/promising one. If the best ranked option is not unique, one could select any of the best-ranked variables. In our implementation, we force determinism by selecting the variable with the lowest index. This way, the serial and the parallel implementations make the same choice, ensuring fairness in comparison.

Let us briefly recall the main families of look-ahead heuristics we are interested in.

Maximum Occurrences in Clauses of Minimum Size

These heuristics [11], briefly referred to as *MOM heuristics*, aim at selecting the unassigned variable that might impact the most in the subsequent unit propagation step, being present in "small" clauses. Variants of the schema appeared in the literature:

1. *Jeroslow-Wang heuristics (JW).* The goal of the JW heuristics is to select variable and value in such a way to maximize the chances of satisfying the formula [12]. This is made by computing the following weight function w for each literal ℓ:

$$\mathsf{w}(\ell) = \sum_{\omega \in \Phi \wedge \ell \in \omega} 2^{-|\omega|} \tag{1}$$

   where Φ is the current set of unresolved clauses and $|\omega|$ denotes the number of unassigned literals in the clause ω. There are two subvariants of JW:

   - *JW-OS (JW One-Sided)* considers the weights $\mathsf{w}(x)$ and $\mathsf{w}(\neg x)$ separately: the VSH selects the unassigned variable x having the largest individual weight (being it $\mathsf{w}(x)$ or $\mathsf{w}(\neg x)$).
   - *JW-TS (JW Two-Sided)* combines the weights $\mathsf{w}(x)$ and $\mathsf{w}(\neg x)$: the VSH selects the unassigned variable x having the largest $|\mathsf{w}(x) - \mathsf{w}(\neg x)|$ value.

   In both cases, the PSH assigns x true, if $\mathsf{w}(x) \geq \mathsf{w}(\neg x)$, false otherwise.
2. *BOHM heuristics* [13]. This heuristics associates to each unassigned variable x an array of weights $\langle \mathsf{w}_1(x), \mathsf{w}_2(x), \ldots, \mathsf{w}_n(x) \rangle$ such that, n is the number of literals in the largest

clause, and for each $i \in \{1, \ldots, n\}$:

$$\mathsf{w}_i(x) = \alpha \cdot \max\Big(lc_i(x), lc_i(\neg x)\Big) + \beta \cdot \min\Big(lc_i(x), lc_i(\neg x)\Big),$$

where $lc_i(\ell)$ denotes the number of occurrences of the literal ℓ in unresolved clauses of size i, and α and β are experimentally tuned parameters. The values used in [13] are $\alpha = 1$ and $\beta = 2$. The VSH selects the unassigned variable x having the maximum array (considering the lexicographical ordering). The PSH assigns x true if $\sum_i lc_i(x) \geq \sum_i lc_i(\neg x)$, false otherwise.

3. *PrOpositional SatIsfiability Testbed heuristics (POSIT)* [11]. This heuristics gives higher priority to unassigned variables having a high number of occurrences in the smallest unresolved clauses. This weight function is evaluated for each variable x:

$$\mathsf{w}(x) = lc_{min}(x) \cdot lc_{min}(\neg x) \cdot 2^{\eta} + lc_{min}(x) + lc_{min}(\neg x),$$

where $lc_{min}(\ell)$ denotes the number of occurrences of the literal ℓ in the smallest unresolved clauses, and η is a sufficiently large constant.[1] The VSH selects the unassigned variable x having the largest weight. The PSH assigns x false if $lc_{min}(x) \geq lc_{min}(\neg x)$, true otherwise. The aim of this heuristics is to maximize the effect of the unit propagation step that will follow the choice step.

Literal Count Heuristics

Briefly referred to as *LC heuristics* [14], their purpose is to select the variable whose assignment causes the satisfaction of the largest number of clauses. To this aim, they classify variables according to the number of their occurrences in unresolved clauses. Let $lc(\ell)$ denote the number of occurrences of ℓ in unresolved clauses. There are two main LC heuristics:

1. *Dynamic Largest Individual Sum (DLIS)* [15] that considers the values $lc(x)$ and $lc(\neg x)$ separately: the VSH selects the unassigned variable x having the largest value (being it $lc(x)$ or $lc(\neg x)$).
2. *Dynamic Largest Combined Sum (DLCS)* [14] that selects the unassigned variable x having the largest $lc(x) + lc(\neg x)$ value.

For both DLIS and DLCS, the PSH assigns x true if $lc(x) \geq lc(\neg x)$, false otherwise.

2.3. GPU and CUDA

The CUDA framework [3] is a general-purpose parallel computing platform and programming model that leverages the parallel computing engine of NVIDIA GPUs. It introduces a number of key abstractions that specifies, in particular,

- a hierarchical organization of threads (i.e., execution flows);

[1]According to [11], η should be such that 2^{η} is larger than the number of unresolved clauses, but small enough to avoid overflow in calculation of $\mathsf{w}(x)$. This allows the solver to enforce preference for variables x having a similar number of occurrences of x and $\neg x$.

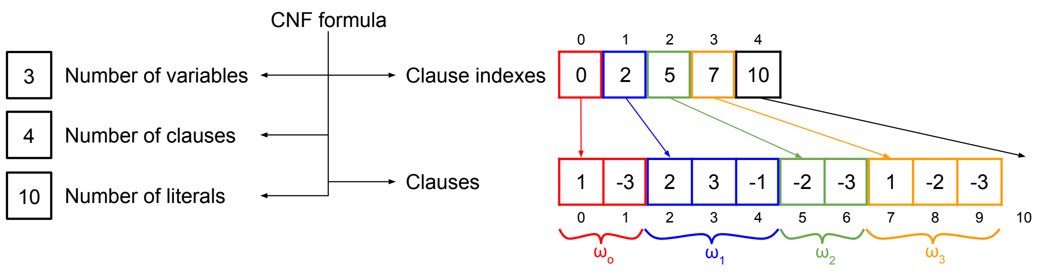


Figure 1: Data structure for the CNF formula $(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$. Each variable x_i is represented by i, minus sign denotes negation.

- a hierarchy of memories (global, shared, constant, local, registers, etc.), with different scopes and lifetimes;
- some synchronization mechanisms.

These abstractions lead the programmer to partition the problem into subproblems that are independently solved in parallel by groups of threads, called *blocks*. In turn, the blocks are organized in *grids*. In particular, CUDA C extends the C language allowing the definition of particular functions, called *kernel*s. Kernels, identified by the use of the keyword `__global__` in their definition, are called by the host (CPU) and ran, in parallel, on multiple threads, on the device (GPU). The desired values of the size of grid and of the blocks per grid are passed as parameter:

```
kernelName<<<GridDim3D, BlockDim3D>>>(Actual Arguments).
```

The NVIDIA GPU architecture consists of thousands of identical compute units, called *cores*, grouped into a uniform collection of *Streaming Multiprocessors (SMs)*. SMs feature a *Single-Instruction Multiple-Thread (SIMT)* execution mode, designed to execute hundreds of threads concurrently. Each SM creates, schedules, and executes blocks of threads, further partitioned in groups of 32 threads, called *warps*. Threads in a warp are intended to execute in lock-step mode. However, each thread has its own program counter and register status. This allows each thread to branch out and execute independently, diverging from the execution of the other threads of its warp. The maximum performance is reached when thread divergence is avoided (so, all threads of a warp execute the same instruction) and when memory accesses patterns are designed to exploit the full bandwidth of each specific kind of memory. We refer the reader to [3] for a detailed presentation of CUDA and to [5, 6, 7] for descriptions of specific implementations of CUDA-based parallel solvers for SAT and ASP.

3. GPU-enhanced Look-ahead Heuristics

We have developed a CUDA C library, called *MiraCle*, implementing the look-ahead heuristics described in Section 2.2. For comparison purposes, the CPU versions have also been implemented using exactly the same data structures.

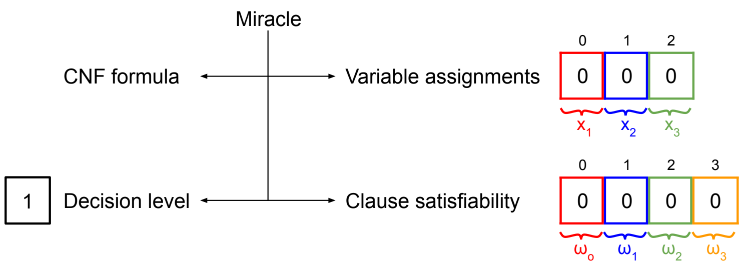


Figure 2: `Miracle` data structure `d_mrc` at (the starting) decision level 1. All variables are unassigned and all clauses are unresolved.

Data Structures Employed

A CNF formula (i.e., a set of clauses) is represented by the data structure depicted in Figure 1. It stores the numbers of variables, clauses, and literals, as well as a *clause array* and a *clause indexing array*. Each literal is represented by an integer index (negative values represent negative literals). Each clause is stored as the sequence of the indices representing its literals. Clauses are stored consecutively in the clause array. Their starting positions are recorded in the clause indexing array. Such a representation of a CNF formula is part of a larger data structure (called `Miracle` and instantiated as `d_mrc` in Figure 2 and in Table 1) which represents the state of the search in a specific moment in time. This data structure encompasses these parts:

- the CNF formula (as described in Figure 1);
- the current decision level (the starting decision level is 1);
- an array va storing the current variable assignments as signed integer values: for each variable i, $va[i] = \ell$ means that i has been assigned at level $|\ell|$ with polarity corresponding to the sign of ℓ. For unassigned variables we set $va[i] = 0$;
- the array cs storing information about clause satisfiability: $cs[j] = \ell$ if ω_j has been satisfied at decision level ℓ, or $cs[j] = 0$ if ω_j is unresolved.

MiraCle Implementation Details

The library provides functionalities that can be partitioned into three parts:

1. initialization and removal of the formula;
2. updating and restoring the formula;
3. computation of look-ahead heuristics.

We list the procedures of the three groups in Table 1. Due to space limitation, we cannot go into detail on all of them here. The interested reader can access the source code in http://clp.dimi.uniud.it/sw/, where they are documented. We restrict our presentation to those of the third group (see also the bottom part of Table 1). In particular, let us focus on the *JW One-Sided* heuristics, since the others essentially differ in the evaluation of the weighting function used to rank literals (cf., Section 2.2). This heuristics is computed by the function `mrc_gpu_JW_OS_heuristics()` (a simplified version of it is shown in Alg. 1 and illustrated in

`Miracle mrc_create_miracle(filename)`	Import a SAT instance in CNF format into the host data structure. Return a pointer to host memory.
`Miracle mrc_gpu_transfer_miracle_host_to_dev(mrc)`	Copy the data structure to the device. Return a pointer to device memory.

`mrc_destroy_miracle(mrc)`
`mrc_gpu_destroy_miracle(d_mrc)`
	Destroy host-side and device-side data structure, respectively.

`mrc_gpu_increase_decision_level(d_mrc)`	Increase the decision level (called before `mrc_gpu_assign_lits()`).
`mrc_gpu_assign_lits(lits, lits_len, d_mrc)`	Assign the literals `lits` true and update the data structure on the device.
`mrc_gpu_backjump(dl, d_mrc)`	Backjump to the decision level `dl` and update the device data structure.

`Lit mrc_gpu_JW_OS_heuristics(d_mrc)`
`Lit mrc_gpu_JW_TS_heuristics(d_mrc)`
`Lit mrc_gpu_BOHM_heuristics(d_mrc, a, b)`
`Lit mrc_gpu_POSIT_heuristics(d_mrc, n)`
`Lit mrc_gpu_DLIS_heuristics(d_mrc)`
`Lit mrc_gpu_DLCS_heuristics(d_mrc)`
	Compute heuristics on the device, w.r.t. the current assignment stored in `d_mrc` (`a`,`b`,`n` are the parameters α, β, η described in Section 2.2). Return the “best” literal.

Table 1
Main components of the *MiraCle* library (see Section 3).

Figure 3). The computation proceeds as follows. After resetting the working array `lit_weights` in global memory and configuring the launch parameters (w.r.t. the number of available SMs), the kernel `JW_weigh_lits_unres_clauses_krn()` is run. In its grid, each thread processes one or more clauses by adopting a grid-stride loop. For each unresolved clause, in lines 16–21, its contribute to the weight of each of its unassigned literals is computed according to (1). The global weights are updated using an atomic instruction for all unassigned literals (lines 22-24) to avoid race conditions. The best selectable literal and its weight are singled out by reducing the array `lit_weights`. The computation of the logarithmic reduction is performed on the GPU by `find_idx_max_float()`, called in line 7 (where, for readability, we improperly used a compact form to denote the retrieval of the result).

Integration Into a SAT Solver

The library has been designed by abstracting from CUDA implementation details and to be easily integrable into any DPLL or CDCL solver. Hence, its use does not require deep CUDA programming skills. Table 1 lists the basic functions that can be used to enhance a (serial) SAT solver with CUDA-based evaluation of look-ahead heuristics. To this aim, it suffices adding suitable calls into the code of the solver. More specifically, the solver has to first initialize the `Miracle` data structure by calling `mrc_create_miracle()`. This creates a representation of the SAT problem (cf., Figure 2) on host (called `mrc` in Table 1). Then, `mrc` is copied to device global memory, using `mrc_gpu_transfer_miracle_host_to_dev()`. This function returns a reference `d_mrc` to the device-side structure, that will be used by all subsequent calls to the library functions. Once `d_mrc` has been created, the solver can proceed as its original algorithm dictates, but each time the solver assigns a literal, increases the current decision level, and consequently performs some propagations, it has to update the device-side structure by calling `mrc_gpu_increase_decision_level()` and `mrc_gpu_assign_lits()` (the

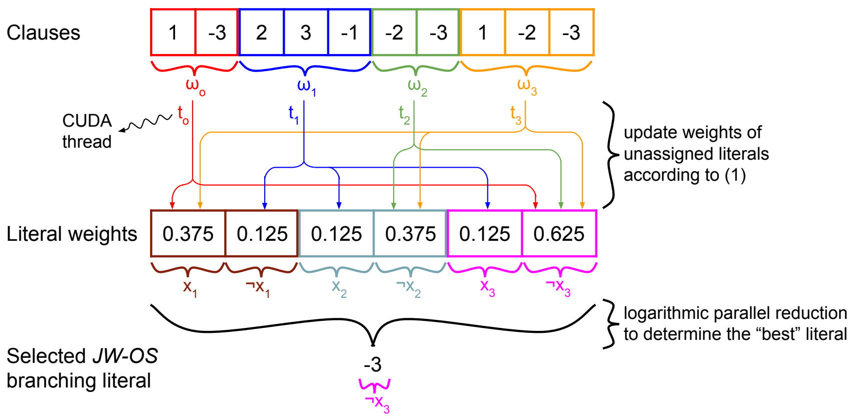

Clauses
1
-3
2
3
-1
-2
-3
1
-2
-3
ω0
ω1
ω2
ω3
CUDA thread
t0
t1
t2
t3
update weights of unassigned literals according to (1)
Literal weights
0.375
0.125
0.125
0.375
0.125
0.625
x1
¬x1
x2
¬x2
x3
¬x3
logarithmic parallel reduction to determine the "best" literal
Selected JW-OS branching literal
-3
¬x3

Figure 3: Computation scheme of the *JW One-Sided* heuristics on the GPU. First, each thread processes an unresolved clause and updates the weight of each unassigned literals in it according to (1). Then, the best selectable literal is computed by reducing the array of literal weights.

latter requires the list of assigned and propagated literals as argument). Similarly, whenever the solver performs a backjump to a level `dl`, the function `mrc_gpu_backjump()` should be used to update `d_mrc`. Each time the solver needs the GPU-based computation of one or more heuristics, the corresponding functions (listed in the bottom part of Table 1) can be called.

4. Experimental Results

To experiment with the library, we integrated it into two existing SAT solvers:

- *SATSolverDPLL*: a DPLL solver developed by Sukrut Rao [16]. This is a basic SAT solver implementing the raw DPLL algorithm. We have chosen this minimal implementation because its essentiality guarantees a fairer comparison between the various branching heuristics (in both their serial and parallel versions), not biased by the effect of other strategies, techniques, and optimizations that are often adopted in implementing a SAT solver.
- *microsat*: a CDCL solver originally developed by Marijn Heule and later modified by Armin Biere [17]. This is a simple conflict-driven SAT solver exploiting watched literals, clause learning, restart, and clause forgetting. It exposes greater performance w.r.t. the aforementioned *SATSolverDPLL* solver. We used this solver to compare the "quality" of the look-ahead heuristics against the lock-back heuristics used by *microsat*, namely VMTF (Variable Move-To-Front [18]).

To experiment with the two GPU-enhanced SAT solvers, we used a server equipped with an octa-core (16 threads) Intel i9-11900K 3.5GHz, with 16 MB cache and 64 GB DRAM, running Ubuntu 20.04.3 LTS (kernel 5.11.0). In this section, we report on experiments ran using a device NVIDIA GeForce RTX 3090 (compute capability 8.6, Ampere architecture, 24 GB, 82 SMs, 10496 CUDA-cores, clock rate 1.7 GHz). The code was compiled using GCC 9.3.0

Algorithm 1: Host and device code for evaluation of the JW-OS heuristics (simplified)

```
static Lit MRC_GPU_JW_OS_HEURISTICS(d_mrc){
    Data: nlits, num_clauses: number of clauses and literals
    Data: lit_weights: array for literal weights (device-side)
    int blks, tpb;
    /* clear lit_weights:                                                    */
    cudaMemset(lit_weights, 0, sizeof(float)*nlits);
    /* retrieve values blks and tpb, computed depending on GPU specs:        */
    configureLaunchParam(num_clauses, &blks, &tpb);
    /* compute literal ranking on the device:                                */
    JW_weigh_lits_unres_clauses_krn<<<blks, tpb>>>(d_mrc);
    /* logarithmic parallel reduction to determine the "best" literal:       */
    Lidx blidx;                       /* Selected JW-OS branching literal index */
    float lw_blidx;                                       /* and its weight */
    /* retrieval of the result:                                              */
    (blidx,lw_blidx) = find_idx_max_float(lit_weights);
    return ((lw_blidx == 0)?UNDEF_LIT:lidx_to_lit(blidx));
}

__global__ void JW_WEIGH_LITS_UNRES_CLAUSES_KRN(d_mrc){
    Data: ncls: number of clauses
    float W;                                          /* weight of a literal */
    Lidx lidx;                               /* index of a literal in a clause */
    register int c_size;            /* number of unassigned literals in a clause */
    /* pointers into d_mrc, kept in registers to speed up accesses:         */
    register int * clss = d_mrc->clause_sat;
    register int * vars = d_mrc->var_ass;
    register int K; register int B;

    for(i=threadIdx.x+blockIdx.x*blockDim.x; i<ncls; i+=blockDim.x*gridDim.x){
     /* using a stride-loop each thread processes one or more clauses       */
        if(!(clss[i])){ /* if the clause is unresolved                      */
            c_size = 0; B = cl_idxs[i]; K = cl_idxs[i+1];
            for(int l = B; l < K; l++){ /* count unassigned lits of clause   */
                lidx = cls[l];
                if(!(vars[lidx_to_var(lidx)])) c_size++;
            }
            W = exp2f((float)-c_size);                       /* compute weight */
            for(int l = B; l < K; l++){ /* update weights of unassigned lits */
                lidx = cls[l];
                if(!(vars[lidx_to_var(lidx)])) atomicAdd(&(lit_weights[lidx]),W);
            }
        }
    }
}
```

and CUDA 11.5. We also ran analogous experiments using other GPUs (such as, NVIDIA Tesla K40c and GeForce GTX 1060), obtaining results in line with those we present here.

The first experiment we report on tries to assess the "*quality*" of the heuristics, namely, how much the outcome of different heuristic selection functions affects the overall computation of SAT solvers. To this aim, we considered the number of heuristics calls a solver has to make before reaching a solution of a SAT instance. Intuitively, a lower number of calls suggests that the heuristics better drives the solver to the solution (or to the detection of unsatisfiability). To run this experiment, we used a dataset of instances from [19]. In particular, we considered 180 instances from the benchmarks *aim*, *Beijing*, *blocksworld*, *dubois*, *ii16*, *ii32*, *jnh*, *logistics*, *pigeon-hole*, *pret*, *ssa*. Figure 4 shows the comparison of the look-ahead heuristics and the native look-back

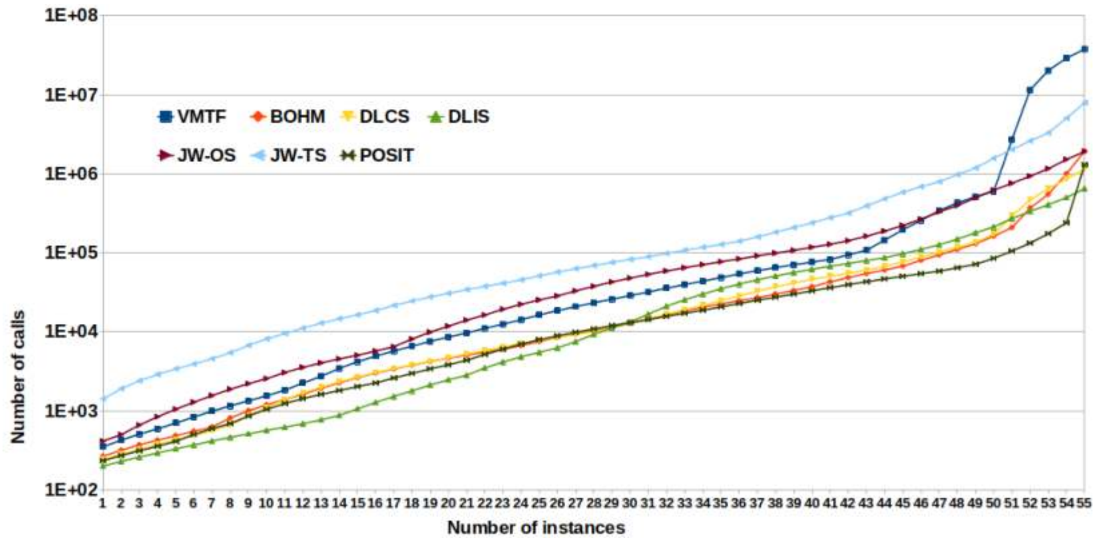


Figure 4: Number of calls to the heuristics needed by *microsat* to solve the instances, using the various look-ahead heuristics and its native look-back heuristics VMTF.

Id	Instance	Size (MB)	Variables	Clauses
I1	`at-least-two-sokoban-sequential-p145-microban-sequential.030-NOTKNOWN.cnf`	45	198252	2385409
I2	`SC21_Timetable_C_557_E_73_Cl_37_S_35.cnf`	63	406207	2841961
I3	`Mycielski-11-hints-4.cnf`	79	15350	3975330
I4	`E00X23.cnf`	104	15364	2133873
I5	`spg_400_281.cnf`	111	792025	4063559
I6	`9dlx_vliw_at_b_iq8.cnf`	161	371419	7170909
I7	`vlsat2_702_14170.cnf`	224	70288	14170788
I8	`13pipe_k.cnf`	239	147626	12295313
I9	`crafted_n12_d6_c4_num17.cnf`	246	56064	15834160
I10	`blocks-blocks-36-0.180-SAT.cnf`	247	733825	13169160
I11	`sokoban-p16.sas.ex.19-sc2016.cnf`	264	2929760	6312685
I12	`barman-pfile10-040.sas.ex.15.cnf`	405	430288	976816
I13	`Kakuro-easy-115-ext.xml.hg_5.cnf`	616	171688	24612456

Table 2
Excerpt of the set of instances used in performance comparison of CPU and GPU implementations of branching heuristics (see Figure 6).

heuristics of *microsat* (VMTF) for an excerpt of those instances whose computation finished within a timeout of 10 minutes, for each of the heuristics we used. The cactus-plot shows the cumulative number of calls (Y-axis) needed to solve a number of instances (X-axis). We observe that the worst performance are those of the two variants of Jeroslow-Wang heuristics, while all the others allow the solver to compute the solution using a significantly lower number of calls to the library functions (observe that the plot uses a log-scale for the Y-axis).

Similar results have been obtained for the solver *SATSolverDPLL*, as can be observed from the analogous cactus-plot shown in Figure 5.

To compare the performance of CPU and GPU implementations of the look-ahead heuristics in

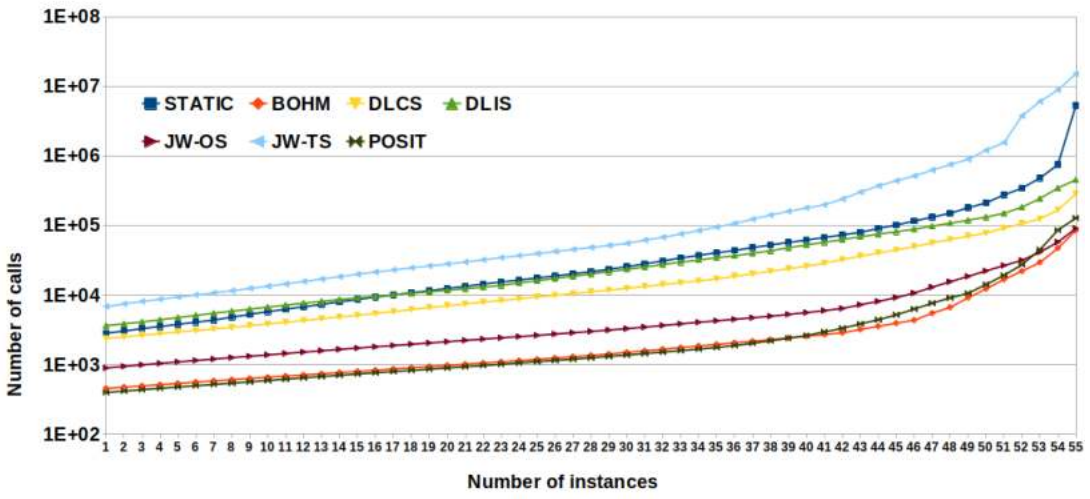


Figure 5: Number of calls to the heuristics needed by *SATSolverDPLL* to solve the instances, using the various look-ahead heuristics and its native heuristics (STATIC, in the chart). Such a native static branching heuristics is the same as DLIS but is executed once for all at the beginning of the computation.

MiraCle, we ran *microsat* on a selection of instances from [20] and evaluated the average time spent in computing each branching literal. We only considered instances having size greater than 40 MB. Figure 6 reports the comparison of the time spent by the 7 different implementations of the BOHM heuristics for a significant excerpt of the dataset (see Table 2). We observe that all parallel implementations outperform the serial one, obtaining 100x average speedup. The best performance are obtained by using 512 threads-per-block: 136x average speedup and 534x maximum speedup. We also notice that the number of threads-per-block significantly influences the performance of the solver in almost all instances. In fact, for each instance, Figure 6 shows how the average time decreases as the number of threads-per-block increases (note that the plot uses a log-scale for the Y-axis). Analogous results have been obtained for the other look-ahead heuristics (charts omitted because of space limits).

The results of the experiments seem to confirm that a GPU-based parallel implementations of a look-ahead heuristics can provide better performance in heuristic function evaluations. Moreover, as expected, the quality of the outcome of these branching heuristics is in general greater than those of the look-back options. Even when this is not the case, as for Jeroslow-Wang (cf., Figs. 4 and 5), the sub-optimal choices in literal selection is compensated by a higher efficiency in heuristics computation (cf., Figure 6).

5. Concluding Remarks

Due to the high computational cost of look-ahead heuristics, designers of modern SAT solvers tend to prefer alternative look-back heuristics. In an attempt to rehabilitate look-ahead heuristics, we described a GPU-based C library, *MiraCle*, implementing CUDA versions of the main

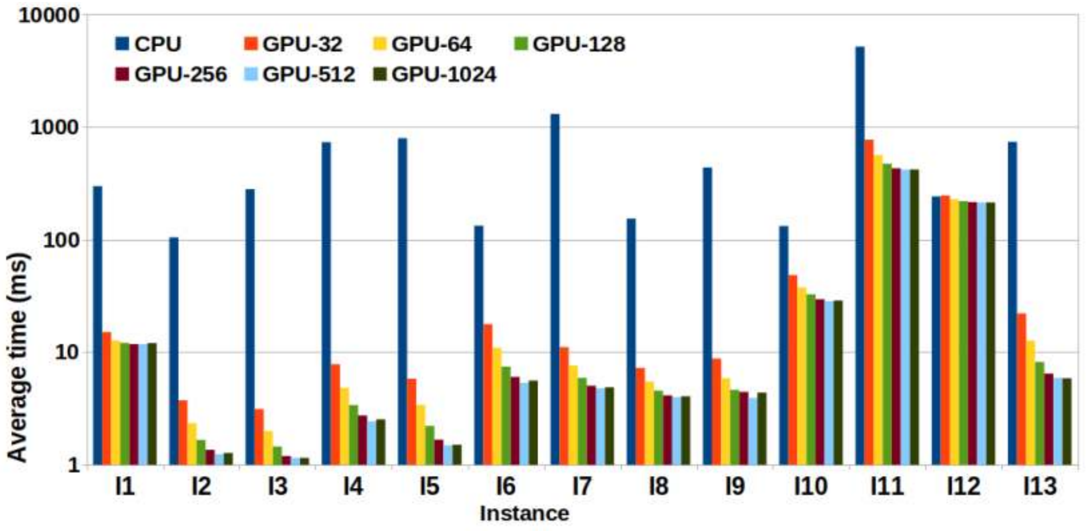


Figure 6: Average time spent in each BOHM heuristics evaluation by the CDCL solver. X-axis: an excerpt of the instances we experimented with. Y-axis: average time in ms. For each instance, we compare the time spent by the CPU implementation and by 6 versions of the CUDA implementation (using 32, 64, 128, 256, 512, and 1024 threads-per-block, resp.).

branching heuristics. We have shown the feasibility of the proposal by realizing the integration of the GPU-based functionalities into two different SAT solvers. Experimentation carried out on a significant number of instances has emphasized how CUDA implementations of the most common look-ahead heuristics can fully exploit the computational power of graphics cards. This allows to enhance a generic SAT solver (not necessarily parallel) by providing it with the parallel computation of such branching heuristics, that, in a purely serial context, would represent an inefficient computational bottleneck. Ameliorations of the library are currently under development. For instance, we plan to improve the implementation of the functionalities offered by *MiraCle* by introducing optimizations that depend on the compute capability of specific GPU in use (e.g., the possibility of exploiting cutting-edge technologies such as *tensor cores* and *warp-level* optimized intrinsic functions). We intend to extend the library by considering other heuristics that appeared in the literature, to investigate whether they might benefit from a parallel implementation. Some examples are the Clause Reduction Heuristics (CRH) [21] proposed in *OKsolver*, the Weighted Binaries Heuristics (WBH) [22] applied in the solver *Satz*, and the Backbone Search Heuristics (BSH) [23]. This would be a first step toward the realization of a purely GPU-based full-blown SAT solver, by means of an integration of *MiraCle* into the existing parallel solvers, such as, for example, the one described in [5].

Other lines of research can benefit from the experience made in designing and improving *MiraCle*. In fact, many of the software solutions designed and implemented to realize a (library supporting) GPU-based SAT solving, have application in the broader field of Computational Logic [24, 25]. We intend to adopt and adapt the approach we described in this paper for SAT, to the prototypical GPU-based solvers we proposed in past research for Answer Set Programming [26, 27] and Constraint Solving [28, 29, 30].

References

[1] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Communications of the ACM 5 (1962) 394–397.

[2] A. Biere, M. Heule, H. van Maaren, T. Walsh, Handbook of Satisfiability, volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2009.

[3] NVIDIA, CUDA C++: Programming Guide (v.11.6), NVIDIA Press, Santa Clara, CA, 2022.

[4] A. Dal Palù, A. Dovier, A. Formisano, E. Pontelli, Exploiting unexploited computing resources for computational logics, in: F. A. Lisi (Ed.), Proc. of the 9th Italian Convention on Computational Logic, volume 857 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2012, pp. 74–88. URL: http://ceur-ws.org/Vol-857/paper_f06.pdf.

[5] A. Dal Palù, A. Dovier, A. Formisano, E. Pontelli, CUD@SAT: SAT solving on GPUs, J. of Experimental & Theoretical Artificial Intelligence (JETAI) 27 (2015) 293–316.

[6] A. Dovier, A. Formisano, E. Pontelli, Parallel answer set programming, in: Y. Hamadi, L. Sais (Eds.), Handbook of Parallel Constraint Reasoning, Springer, 2018, pp. 237–282.

[7] A. Dovier, A. Formisano, F. Vella, GPU-based parallelism for ASP-solving, in: P. Hofstedt, S. Abreu, U. John, H. Kuchen, D. Seipel (Eds.), Revised Selected Papers from DECLARE 2019, volume 12057 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 3–23.

[8] S. A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the third annual ACM symposium on Theory of computing, 1971, pp. 151–158.

[9] L. A. Levin, Universal sequential search problems, Problemy Peredachi Informatsii 9 (1973) 115–116.

[10] M. Davis, H. Putnam, A computing procedure for quantification theory, Journal of the ACM 7 (1960) 201–215.

[11] J. W. Freeman, Improvements to Propositional Satisfiability Search Algorithms, Ph.D. thesis, University of Pennsylvania, 1995.

[12] R. G. Jeroslow, J. Wang, Solving propositional satisfiability problems, Annals of Mathematics and Artificial Intelligence 1 (1990) 167–187.

[13] M. Buro, H. Kleine Büning, Report on a SAT competition, Bulletin of EATCS 49 (1992).

[14] J. P. Marques-Silva, The impact of branching heuristics in propositional satisfiability algorithms, in: Portuguese Conference on Artificial Intelligence, Springer, 1999, pp. 62–74.

[15] J. P. Marques-Silva, K. A. Sakallah, GRASP — a new search algorithm for satisfiability, in: The Best of ICCAD, Springer, 2003, pp. 73–89.

[16] S. Rao, SAT-Solver-DPLL, github.com/sukrutrao/SAT-Solver-DPLL. Last accessed in 2022.

[17] M. Heule, A. Biere, Microsat, github.com/arminbiere/microsat. Last accessed in 2022.

[18] L. Ryan, Efficient Algorithms for Clause-Learning SAT Solvers, Master's thesis, Simon Fraser University, 2004.

[19] H. Hoos, SATLIB - benchmark problems, www.cs.ubc.ca/~hoos/SATLIB/benchm.html, 2022.

[20] M. Heule, M. Järvisalo, M. Suda, M. Iser, T. Balyo, N. Froleyks, SAT Competition 2021, satcompetition.github.io/2021, 2021.

[21] O. Kullmann, Investigating the behaviour of a SAT solver on random formulas, Technical Report CSR 23-2002, University of Wales Swansea, Swansea Wales, UK, 2002

[22] C. M. Li, et al., Look-ahead versus look-back for satisfiability problems, in: International

Conference on Principles and Practice of Constraint Programming, Springer, 1997, pp. 341–355.

[23] O. Dubois, G. Dequen, A backbone-search heuristic for efficient solving of hard 3-SAT formulae, in: IJCAI, volume 1, 2001, pp. 248–253.

[24] G. Gupta, E. Pontelli, K. A. M. Ali, M. Carlsson, M. V. Hermenegildo, Parallel execution of Prolog programs: a survey, ACM Trans. Program. Lang. Syst. 23 (2001) 472–602.

[25] A. Dovier, A. Formisano, G. Gupta, M. V. Hermenegildo, E. Pontelli, R. Rocha, Parallel logic programming: A sequel, Theory and Practice of Logic Programming (2022) 1–69. doi:10.1017/s1471068422000059.

[26] A. Dovier, A. Formisano, E. Pontelli, F. Vella, A GPU implementation of the ASP computation, in: M. Gavanelli, J. H. Reppy (Eds.), PADL 2016, volume 9585 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 30–47.

[27] A. Dovier, A. Formisano, E. Pontelli, F. Vella, Parallel Execution of the ASP Computation, in: M. De Vos, T. Eiter, Y. Lierler, F. Toni (Eds.), Tech.Comm. of ICLP 2015, volume 1433, CEUR-WS.org, 2015.

[28] A. Dovier, A. Formisano, E. Pontelli, F. Tardivo, {CUDA}: Set constraints on GPUs, Rendiconti dell'Istituto di Matematica dell'Università di Trieste 24 (2021).

[29] F. Campeotto, A. Dal Palù, A. Dovier, F. Fioretto, E. Pontelli, Exploring the Use of GPUs in Constraint Solving, in: M. Flatt, H. Guo (Eds.), Proc. of PADL 2014, volume 8324 of *Lecture Notes in Computer Science*, Springer, San Diego, CA, USA, 2014, pp. 152–167. URL: http://dx.doi.org/10.1007/978-3-319-04132-2_11. doi:10.1007/978-3-319-04132-2_11.

[30] F. Tardivo, A. Dovier, A. Formisano, L. Michel, E. Pontelli, Constraints propagation on GPU: A case study for AllDifferent, in: R. Calegari, G. Ciatto, A. Omicini (Eds.), Proc. of the 37th Italian Conference on Computational Logic (CILC 2022), CEUR Workshop Proceedings, CEUR-WS.org, 2022.

Epistemic Multiagent Reasoning with Collaborative Robots

Davide Soldà[1,2,*], Francesco Fabiano[1,3,*] and Agostino Dovier[1,4]

[1]*Constraint and Logic Programming Lab, University of Udine, 33100 Udine, Italy.* http://clp.dimi.uniud.it/
[2]*Embedding System unit of Fondazione Bruno Kessler, 38123 Trento, Italy*
[3]*Department of Mathematical, Physical and Computer Sciences, University of Parma, 43124 Parma, Italy*
[4]*Department of Mathematics, Computer Science and Physics, University of Udine, 33100 Udine, Italy*

Abstract

Over the last few years, the fields of Artificial Intelligence, Robotics and IoT have gained a lot of attention. This increasing interest has brought, among other things, to the development of autonomous multi-agent systems where robotic entities may interact with each other. As for all the other autonomous settings, also these systems require arbitration. Our work tries to address this problem by presenting a framework that embeds both a classical and a multi-agent epistemic (epistemic, for brevity) planner in a robotic control architecture. The idea is to combine the *(i)* classical and the *(ii)* epistemic solvers to model efficiently the interaction with: the *(i)* physical world and the *(ii)* information flows, respectively. In particular, the presented architecture starts by planning on the "epistemic level" refining then single-agent world-altering actions thanks to the classical planner. To further optimize the solving process, we also introduce the concept of macros in epistemic planning. Macros, in fact, have been successfully employed in classical planning as they allow for opportune aggregations of actions that may lead to a reduction of plans' length. Finally, the overall framework is exemplified and validated with two Franka Emika manipulators. This allowed us to empirically justify how the combination of the two planning approaches (classical and epistemic), and the introduction of macros, reduce the computational time required by the orchestrating phase.

Keywords

Multi-Agent Planning, Epistemic Reasoning, Answer Set Programming, Robot Operating System

1. Introduction

In the recent years, the field of cognitive robotics experienced significant progress, both from the academic (see [1] for a detailed survey) and the industrial, *e.g.* [2], [3], [4], point of view. Even if most of current autonomous systems frameworks are designed for scenarios where only a single entity interacts with the environment, also multi-agent settings are well studied [5], [6]. However, there is not focus on modeling robots beliefs in goals or action pre/post conditions.

In particular, we envisioned and developed e-PICO (**EPI**stemic Reasoner **C**ollaborative R**O**bots): a planning framework that coordinates a set of autonomous agents. This setup is able

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
*Corresponding author.

solda.davide@spes.uniud.it (D. Soldà); francesco.fabiano@unipr.it (F. Fabiano); agostino.dovier@uniud.it (A. Dovier)
0000-0002-1161-0336 (F. Fabiano); 0000-0003-2052-8593 (A. Dovier)

to reason on complex multi-agent epistemic concepts while dealing with their computational issues, that arise from the inherent complexity of epistemic reasoning. This is achieved by combining techniques from the Multi-agent Epistemic Planning (MEP) field and the more efficient classical planning approaches. Thanks to this combination our architecture can benefit from the generality derived by MEP solvers and the efficiency of the classical ones. That is, `e-PICO` consider both of an epistemic and a classical description of the planning problem and combines the two solving techniques in a hierarchical way. As we will see later (Section 4) in much greater detail, the architecture firstly solves the problem with a greater level of abstraction using the epistemic solver. Then, the classical resolution process is used to refine the world-altering actions suggested by the epistemic counterpart.

Finally, inspired by [7], we also propose the concept of *macros* in the MEP environment. The use of macros helps in reducing the burden of the epistemic planning process, allowing `e-PICO` to address multi-agent robotics planning scenarios. In fact, such domains present several complications, *e.g.*, the number of possible actions, that if not addressed correctly could render the solving process unfeasible.

To the best of our knowledge, both *i)* the combination of the epistemic and a classical solving processes to model a robotic environment with epistemic goals; and *ii)* the use of macros in the multi-agent epistemic planning context are original contributions of this work.

2. Action Languages and Planning

The area of *automated planning* is one of the most prominent in Artificial Intelligence. This branch studies how to devise tools that help us in deciding the best course of actions to reach a given goal, an activity that is done continuously in our life. Automated planning, therefore, represents one of the most interesting aspect of AI and, consequently, has been vastly studied [8, 9, 10]. Even if we often need to make decisions based on our beliefs, about the environments and about others' beliefs, automated planners usually do not consider such intricacy. In fact, most of the efforts in the planning community address domains where the concept of beliefs and/or knowledge is not taken into account. Nonetheless, the growing interest in AI is pushing researchers to steadily improve and to model more realistic scenarios. This momentum brought, among other things, to the formalization of a far more compelling (w.r.t. more classical approaches) form of planning, that is the so-called *Epistemic Multi-Agent Planning* (MEP). This area of planning reasons within environments where the streams of "knowledge" or "beliefs" need to be considered (see [11, 12] for a more detailed introduction to the topic).

2.1. Multi-Agent Epistemic Planning

Formalizing and reasoning on the idea of knowledge and beliefs has always been of great interest among various research fields (*e.g.*, philosophy, logics, and computer science). In particular, in 1962, Hintikka proposed the first complete axiomatization of these concepts [13]. From this initial effort stemmed the field of *epistemic/doxastic logic* which aims to formalize and reason on information. While this area of research presents various challenges, it focused only on capturing the knowledge relations in static domains. To represent even more interesting and realistic scenarios *Dynamic Epistemic Logic* (DEL) was introduced; that is the logic of reasoning on information flows in dynamic domains where agents can act and alter these relations themselves.

DEL represents the foundation of Multi-Agent Epistemic Planning, the setting concerned with finding the best series of action, that modifies the information flows, to reach goals that (might) refer to agents' knowledge/beliefs. In what follows, for brevity, we will use the term "knowledge" to encapsulate *both* the notions of an agent's knowledge and beliefs. In fact, these concepts are captured by the same modal operator in DEL and their difference resides in structural properties that the epistemic states respect (see [11] for more details). As it is not the objective of this paper to completely present MEP, in what follows we will provide only some fundamental concepts that are necessary to explain the contribution of this work. Far more complete introductions to this topic may be found in [11, 14, 15].

Let us start by presenting the language of well-formed DEL formulae used to express agents' knowledge. This is expressed as follows:

$$\varphi ::= \mathsf{f} \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathbf{B}_{\mathsf{i}}\varphi \mid \mathbf{C}_{\alpha}\varphi,$$

where f is a propositional atom called a *fluent*, i is an agent that belongs to the set of agents $\mathcal{AG}$ s.t. $|\mathcal{AG}| \geq 1$, φ and ψ are belief formulae and $\emptyset \neq \alpha \subseteq \mathcal{AG}$. A *fluent formula* is a DEL formula with no occurrences of modal operators. A *belief formula* is recursively defined as follows:

- A fluent formula is a belief formula;
- If φ is a belief formula and $\mathsf{i} \in \mathcal{AG}$, then $\mathbf{B}_{\mathsf{i}}\varphi$ ("i knows/believes that φ") is a belief formula where the modal operator $\mathbf{B}$ captures the concept of *knowledge*;
- If φ_1, φ_2 and φ_3 are belief formulae, then $\neg\varphi_3$ and $\varphi_1 \,\mathtt{op}\, \varphi_2$ are belief formulae, where $\mathtt{op} \in \{\wedge, \vee, \Rightarrow\}$;
- If φ is a belief formula and $\emptyset \neq \alpha \subseteq \mathcal{AG}$ then $\mathbf{C}_{\alpha}\varphi$ is a belief formula where $\mathbf{C}_{\alpha}$ captures the *Common knowledge* of the set of agents α.

The formula $\mathbf{C}_{\alpha}\varphi$ translates intuitively into the conjunction of the following belief formulae:

- every agent in α knows φ;
- every agent in α knows that every agent in α knows φ;
- and so on *ad infinitum*.

The semantics of DEL formulae is traditionally expressed using *pointed Kripke structures* [16], but also other representations are possible [17, 18]. We refer the interested reader to [11, 12, 17] for a comprehensive introduction to how Kripke structures, or similar formalisms, are used to capture the idea of an epistemic state and how the concept of entailment is defined.

Let us note that the epistemic action language that we will consider in our work implements three *types of action* and three *observability relations*. These are standard concepts in the epistemic planning community and, therefore, we will provide an intuitive description of those addressing the interested reader to [14] for a complete description of this topic. In particular, we assume that each agent can execute one of the following types of action:

- *World-altering* action (also called *ontic*): used to modify certain properties (*i.e.*, fluents) of the world.
- *Sensing* action: used by an agent to refine her beliefs about the world.
- *Announcement* action: used by an agent to affect the beliefs of other agents.

Moreover, each agent is associated to one of the following observability relations during an action execution:

- *Fully-observant*: the agent is aware of the action execution and also knows the effect of the action.
- *Partially-observant*: the agent is aware of the action execution without knowing the effects of the action. Let us note that no agent can be partially observant of an ontic action as it is impossible to decouple the witnessing of a world-altering action and the witnessing of its effects.
- *Oblivious*: the agent is not even aware of the action execution.

Each type of action defines a transition function and alter an epistemic state in different ways. Given the complexity of the topic and the space limitations we address the reader to [14, 18, 19] for a formal definition of these, and others, update functions on diverse epistemic state representations.

Finally, let us introduce the concept of *MEP domain* that, intuitively, contains the information needed to describe a planning problem in a multi-agent epistemic setting.

Definition 2.1 (MEP Domain). *A* multi-agent epistemic planning domain *is a tuple* $\mathcal{D} = \langle \mathcal{F}, \mathcal{AG}, \mathcal{A}, \varphi_{ini}, \varphi_{goal} \rangle$*, where* $\mathcal{F}$*,* $\mathcal{AG}$*,* $\mathcal{A}$ *are the sets of* fluents*,* agents*,* actions *of* $\mathcal{D}$*, respectively;* φ_{ini} *and* φ_{goal} *are DEL formulae that must be entailed by the* initial *and* goal e-state*, respectively. The former e-state describes the domain's initial configuration while the latter encodes the desired one.*

We refer to the elements of a domain $\mathcal{D}$ with the parenthesis operator; *e.g.*, the fluent set of $\mathcal{D}$ is denoted by $\mathcal{D}(\mathcal{F})$. An *action instance* $\mathsf{a}\langle\mathsf{i}\rangle \in \mathcal{D}(\mathcal{AI}) = \mathcal{D}(\mathcal{A}) \times \mathcal{D}(\mathcal{AG})$ identifies the execution of action a by an agent i. Let $\mathcal{D}(\mathcal{S})$ be the set of all possible e-states of the domain. The *transition function* $\Phi : \mathcal{D}(\mathcal{AI}) \times \mathcal{D}(\mathcal{S}) \to \mathcal{D}(\mathcal{S}) \cup \{\emptyset\}$ formalizes the semantics of action instances (the result is the empty set if the action instance is not executable).

2.2. The $\mathcal{A}^V$ language

In this section we briefly introduce the $\mathcal{A}^V$ action description language, which has been designed to be integrated into e-PICO. It essentially corresponds to language $\mathcal{A}$, but some typed variables have been introduced (see [20] as a reference for the nomenclature of planning languages). Typed variables are merely used to write schemes describing finite sets of causal laws that are formed according to the same pattern. Schemes are particularly useful in some application domains, such as when an action that models an agent's movement can be parameterized by a variable that can be instantiated with a series of different positions.

In `e-PICO` the $\mathcal{A}^V$ planning instances are translated into an Answer Set Programming ASP instances. ASP is one the most prominent logic programming paradigms and it is particularly useful in knowledge-intensive applications, see Gelfond and Lifschitz [21] for an introduction to its semantic. Its use as a planning framework, has been deeply explained in [22]. For space reasons we omit the correctness proof of the compiler from $\mathcal{A}^V$ to ASP.

The choice of encoding the planning problems in ASP, already presented in [23], is justified by the fact that ASP as a planning engine outperforms standard task planner for PDDL, where *i)* domains are rich in fluents and *ii)* plans are usually short [24]. Such characteristics precisely characterize how we want to use the classical planner in the robotics domain use case.

3. Collaborative Robots

3.1. Robot Operating System

Robot Operating System (ROS) is the standard *de facto* framework for developing robotic applications. It consists of a communication framework for sending messages between processes. The ROS integrated executable programs are instantiated as special processes, known as nodes, and are organized in packages. A ROS package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. There are several packages which are directly provided by ROS and which can be used by user-defined programs. One package that is particularly used in this work is `MoveIt!`, which is widely used for motion planning and execution in the robotics community.

3.2. The Robots Franka Emika

The framework here presented has been validated in a simple multi-agent scenario in which two Franka Emika robots have been involved. Franka Emika is a robotic arm with 7 Degrees Of Freedom (DOF) with torque sensors at each joint. It is also equipped with a gripper that allows the automated “arm" to handle objects.

The client side of the Franka Control Interface is called `libfranka`. At a higher level we find `franka_ros`, which is a ROS package that contains the description of the robot and the end-effector in terms of kinematics, joint limits, visual surfaces and collision space, an hardware abstraction of the robot for the ROS control framework based on the `libfranka` API, and a set of services to expose the full `libfranka` API in the ROS ecosystem. Let us note that, although the target robots are two Franka Emika, the architecture can be adapted to another collection of manipulators with only minor changes.

4. The e-PICO System

4.1. Macros in Multi-Agent Epistemic Planning

As already mentioned, a significant contribution of this work is the formalization, and consequent employment, of *macros* in the MEP setting. A *macro*, that can be informally described as "an encapsulated sequences of elementary planning operators", is formally defined as follows:

Definition 4.1 (Macro). *Let $\mathcal{D}(\mathsf{i}), \mathcal{D}(\mathsf{j}) \in \mathcal{D}(\mathcal{AI})$ be two action instances, and $\mathsf{s} \in \mathcal{D}(\mathcal{AG})$ be an e-state of a given a domain $\mathcal{D}$. A macro $\mathsf{m}_{\mathsf{i},\mathsf{j}} \in \mathcal{D}$ representing the subsequent execution of j after i can be defined as $\Phi(\mathsf{j}, \Phi(\mathsf{i}, \mathsf{s}))$. Let us note that* i) *we assume that if action a is not executable in a state s, then the result of the update $\Phi(\mathsf{a}, \mathsf{s})$ is $\perp$; and* ii) *the execution of any action a over $\perp$ results in $\perp$ as well.*

The introduction of macros is justified by the fact that, often, patterns of actions performed in sequence are repeated in the same domain. For example, it often happens that an ontic action is followed by an announcement action, because an agent may want to communicate the results of the former to another (oblivious) agent. For now, as pointed out in the Dagstuhl seminar [25], there have been many proposals in the classification of epistemic actions. As pointed out above, the community mostly agrees that the basic classification considers *ontic*, *announcement* and *sensing* as possible categories. Among these three types of actions, only the ontic one has some effect on the physical world. That is why we consider the renaming two, *i.e.*, sensing and announcement, as *purely epistemic* actions. Consequently, we say that a task is purely epistemic if it involves only announcement or sensing actions, or a macro aggregating these two type. In Listing 1 the function call `is_pure_epistemic_task(task)` checks exactly this property.

4.2. The Architecture

In this section, we describe the general architecture and functionality of the e-PICO system, that is the main contribution of this work. e-PICO provides high-level knowledge representation and planning capabilities to ROS-based autonomous robots via the action description languages $\mathcal{A}^V$ (Section 2) and E-PDDL [26]. The main object of our framework is to provide a tool that supports multi-agent epistemic planning in the robotics setting. This is accomplished through a combination of MEP and classical planning solving techniques. In particular, e-PICO combines these two strategies in a hierarchical way. To be more precise, our architecture firstly employs multi-agent epistemic planning for generating tasks, then, at lower level, it exploits a classical planner to break down tasks into simpler actions. Finally, e-PICO directly converts the results of the planning processes into `MoveIt!` commands that can be then executed by the robots. Let us note that we employ an hierarchical combination of the two solving techniques (*i.e.*, classical and epistemic) to avoid unfeasibility in the planning process for domains rich in fluents and actions. The key idea is, in fact, to abstract most of the domain intricacy from the epistemic level and handle it at the classical level, when it is possible, given its vastly superior performances.

Let us now explain in greater detail how the e-PICO system processes, solves and executes a planning problem in the robotic environment. To better visualize the overall framework, we present a graphical representation of it in Figure 1. First of all the *master process* reads the domain descriptions, that contain the initial state description for both the MEP and classical

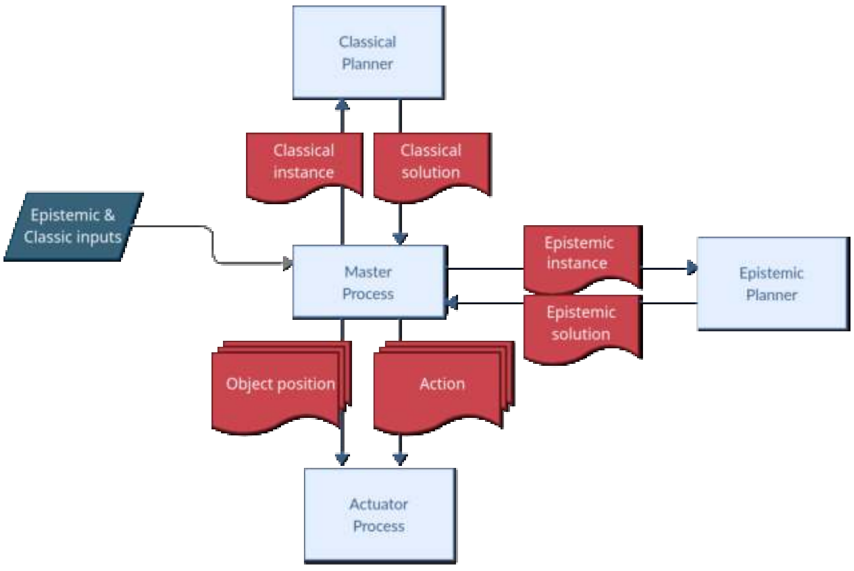


Figure 1: A class-based representation of the e-PICO system.

setting. At this point it is able to send messages to the *actuator process* regarding the objects that need to be considered by the motion planner in order to avoid collisions. Once these messages arrive to the *actuator process*, it adds them to the `MoveIt!` planning scene.

Then, it sends the epistemic planning instance, specified in E-PDDL [26], to an epistemic planner, *i.e.*, EFP [18], which is employed as a black box and returns the sequence of epistemic tasks to be executed. After this first resolution, each task is properly processed following Listing 1. The first step is to break down (*to flat*) macros, then we can reason on a sequence of no macro tasks, hereinafter called *simple tasks*. Intuitively, a simple pure epistemic task can be directly executed. Simple ontic tasks that alter the physical world will undergo further processing. To break down ontic simple tasks into a sequence of classical actions, the *master process* defines an instance of classical planning where the initial and the goal states are defined to match the ontic simple task's conditions and effects. Then, iteratively, the *master process* sends the classical action tasks to the *actuator process*, which translate them into `MoveIt!` commands, and makes finally the robots execute the movement.

Listing 1: pseudo-code for the main steps of e-PICO.

```
def process_e_plan(list<task> task_plan, c_init_state c_init):
  for task in task_plan:
    if is_macro_task(task):
      simple_task_plan = break_down_macro(macro=task)
      for simple_task in simple_task_plan:
        c_init = process_simple_task(t, c_init)
    else:
      c_init = process_simple_task(task)
```

```
def process_simple_task(task t, c_init_state c_init):
  # invariant: not is_macro_task(t)
  if is_epistemic_task(task=t): # sensing or announcement
    execute(pure_epistemic_task=t)
  else:
    c_plan, c_init = send_to_c_planner(ontic_task=t, initial_state=
        c_init)
    for c_action in c_plan:
      send_to_actuator(c_action)
  return c_init
```

4.3. Modeling the Problem Instance

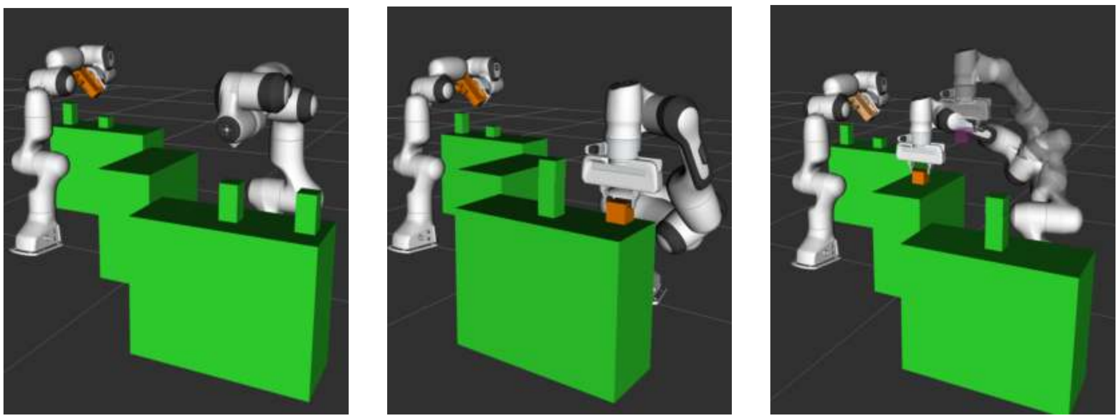
When we model a scenario as a planning problem in order to eventually effectively execute it, we need to address the "anchoring" problem. I.e., we need to connect the model description to the physical objects in the real world. Furthermore, in the `e-PICO` approach, the user also needs to choose what should be modelled at the epistemic level and what at the classical one, the latter in fact should provide low-level actions that are easy to interface with the `MoveIt!` API. Let us take the process of grasping, executed by an automated arm, as an example. It is composed of different phases: i) *pre-grasping*, in which the *end-effector* approaches the object to be grasped with a given direction and orientation. ii) actual *grasping*, in which the final joints of the gripper close. Finally, iii) *post-grasping*, in which the end-effector moves away from the position in which it grasped the object in a given direction and orientation. During each of these phases process we must make sure that the robotic arm does not collide with the objects in its workspace. Therefore, `MoveIt!` is provided with information about location of fixed objects to calculate a trajectory without collisions with fast algorithms, such as *Rapidly-exploring Random Tree*. Furthermore, the aspects of pre-grasping and post-grasping are hidden in the planning model, that sees the grasping action as atomic, the control of the three-phases of grasping has been hard-coded in the *actuator process*.

4.4. Case study

We can now present an application of `e-PICO` in a two agents scenario. The diverse problem instances contain several fluents and actions, as usual in the robotics domain.

The two agents involved are two robotic arms, called `robot1` and `robot2`. In front of each of them, three stacks of colored blocks lie on a table. Initially, the they do not know what blocks they are able to manipulate (Figure 2a presents an example of initial state). To collect such information, they can perform sensing actions to see which blocks they initially have in front of them. Robots can also take a block (as depicted in Figure 2b) to then place it on the *shared table*, as in Figure 2c. We assume that only once a robot has placed the block on the shared table, it is able to communicate the color of such a block to the other arm. Examples in Figure 2 are obtained using the RViz simulator [27] integrated in `MoveIt!`.

For the sake of readability, we will not report all the actions' descriptions, rather we will show only some meaningful examples. Initial state and the fluents encoding are not presented, because they just follow the standard PDDL notation [28, 29]. Let us start, by showing an ontic



(a) Initial state. (b) A robot picking up a block from its table. (c) A robot placing a block on the shared table.

Figure 2: Examples of the robotic test environment states. A video of the "real" execution with the Panda Robots is available: http://clp.dimi.uniud.it/sw/

action in Listing 2. This action encodes the idea that if an agent beliefs that it is holding a block of a certain color and the shared table is free, then, after the execution, the agent will occupy the shared table with the aforementioned block (and will know about this). For a detailed explanantion on the syntax of E-PDDL we address the reader to [26]

Listing 2: Example of ontic action.

```
(
 :action        pick_and_place_on_shared
 :act_type      ontic
 :parameters    (?ag ?ag2 - agent ?c ?c2 - color)
 :precondition  (and  ([?ag](hold ?ag ?c)) (not (occup_by ?ag2 ?c2)))
 :effect        (and (occup_by ?ag ?c) (hold ?ag ?c))
 :observers     (?ag)
)
```

Similarly, for instance, an annoucement action can be executed by the agent that has just moved the block, announcing the newly verified property to the other arm.

Listing 3: Example of annoucement action.

```
(
 :action        announce_block_on_barrier
 :act_type      announcement
 :parameters    (?ag - agent)
 :precondition  (and (occup_by ?ag ?c) (hold ?ag ?c) ([?ag](occup_by ?
    ag ?c)) ([?ag](hold ?ag ?c)))
 :effect        (and (occup_by ?ag ?c) (hold ?ag ?c))
 :observers     (forall (?ag2 - agent) (?ag2))
)
```

We can now apply the 4.1 definition to actions Listings 2 and 3 with the following macro as result, see Listing 4.

Listing 4: Example of a *macro* aggregating an ontic and an announcing action.

```
(
 :action        announce_pick_and_place_on_shared
 :act_type      ontic
 :parameters    (?ag ?ag2 - agent ?c ?c2 - color)
 :precondition  (and   ([?ag](hold ?ag ?c)) (not (occup_by ?ag2 ?c2)))
 :effect        (and (occup_by ?ag ?c) (hold ?ag ?c))
 :observers     (forall (?ag2 - agent) (?ag2))
)
```

Let us now consider what happens to an epistemic ontic action that is refined by the classical planner. Consider a configuration in which the a red block is stacked under a black block and the epistemic ontic action suggests to move the red block on the shared table. The robot cannot take directly the red block, but it should firstly move the black one at the top of another stack and then take the red block to put it on the shared table. Therefore such an epistemic ontic task may be translated into the following sequence of actions:

Listing 5: List of single-arm actions.

```
% pick the black block over the red one from stack 2, pos 1
pick(black,red,1,2)

% place the black block over the green one in stack 1, pos 2
place(black,green,2,1)

% pick the red block from stack 2,pos 0
pick(red,table,0,2)

% place the red block on the shared table
put_on_barrier(red)
```

5. Experimental results

To further asses and demonstrate the capabilities of the developed framework, we tested our architecture on a wide spectrum of scenarios that stem from the configuration described in Section 4.4. Experiments showed the positive impact of using macros in the MEP setting. Furthermore, we were also able to prove the feasibility of the solving procedure in real-world situations.

All the experiments have been conducted on Intel i7-8565U CPU at 1.80GHz and Ubuntu 18.04 OS. For each problem instance, a time limit of 110 seconds was applied. Clingo [30] was used to solve the ASP encoding of $\mathcal{A}^V$, and EFP [18] to tackle the resolution of MEP problems.

First let us highlight how the use of macros can help in reducing the solving time, in Table 1. Test-cases are obtained with a cross product between goal with an increasing difficulty and with an increasing number of blocks inserted in the domain (rows and columns of Table 1 respectively). In particular, we have that:

α: has the objective to let one agent know the color of at least one block initially located in the table in front of the other robot;

β: has same goal as of α while also requesting that the shared table must be free at the end of the plan;

γ: encodes the scenario where the goal is for a robot to know two different colors of blocks initially placed in front of the other arm;

δ: shares the goal of γ with an "extra" condition imposing that the shared table must be free when the planning process is concluded.

For the sake of readability we will make use of the following notations in Table 1:

- ND, that stands for Not Defined. This is used, for example, to indicate that instance γ cannot be performed with just one color as it required that a robot learns two different colors while only one is available.

- TO stands for Time Out. As said before, after 110 seconds the planning procedure is forcefully stopped if it could not find a solution.

From the domain it is evident that whenever an agent takes a block from the shared table, to place it on top of one of its stacks, announcing that the table has been freed right after is very convenient. This small sequence of actions constitutes the macro `announce_pick_and_place_on_shared`. Finally, in Table 1, we compare the solving process when this macro is not activated ("Macro no") and when it is ("Macro yes"). In summary, it is evident that macros improve considerably performance. Their use makes it possible to have better scalability both in terms of number of fluents and of plan lengths.

	1 Color		**2** Colors		**3** Colors		**4** Colors	
Macro	no	yes	no	yes	no	yes	no	yes
α	0.019	**0.004**	1.200	**0.080**	5.192	**1.382**	104.000	**25.889**
β	0.083	**0.013**	4.946	**0.242**	21.780	**4.235**	TO	**85.144**
γ	ND	ND	18.663	**1.427**	TO	**24.805**	TO	TO
δ	ND	ND	TO	**4.827**	TO	**92.998**	TO	TO

Table 1
Time, in seconds, to find a goal, given an initial state. Each instance varies the number of available colors. In **boldface** is highlighted the fastest solving process w.r.t. the activation of the *pspb_and_announce* macro. Let us note that all the values were obtained by averaging the times of 5 iterations on the same instance.

For the sake of completeness, let us now address the initial state generation. It is the first step performed during the calculation of the espitemic plan. This process is empirically almost independent of the use of macros, but it is affected by the number of fluents and actions used in the planning model. To better exemplify this, let us report the average times needed to compute the initial states, increasing the number of fluents, in our case the number of colored blocks: **1)** for one color 0.003 s; **2)** for two colors 0.031 s; **3)** for three colors 0.126 s; and **4)** for four colors 2.117 s. As a consequence of this observation, it is a good rule of thumb, to limit the number of fluents considered to the strict necessary required for the epistemic planning.

Finally, as a design choice we decided to run the classical planner at run-time, each time its use is required to break down an ontic task in a sub-plan of classical actions. Justification for this choice is provided by Table 2, in which, for each instance, the cumulative time required by the A^V solver and the times it was called are reported. Let us note that we did not report the times required by the epistemic planner without the support of the classical one, *i.e.*, the times required by the solver if the domain was entirely described at the epistemic level. The reason is that, without the assistance of the "breakdown" procedure, provided by the classical solver, the solving time always reached the timeout.

	1 Color		**2** Colors		**3** Colors		**4** Colors	
	time	calls	time	calls	time	calls	time	calls
α	0.074	1	0.083	1	0.092	1	0.112	1
β	0.157	2	0.172	2	0.177	2	0.202	2
γ	ND	ND	0.250	3	0.267	3	TO	TO
δ	ND	ND	0.332	4	0.351	4	TO	TO

Table 2
Cumulative time, in seconds, to break down ontic tasks into classical actions. The number of calls to the classical planner is reported in the column "calls".

6. Related Work and Conclusions

While the field of cognitive robotics has made significant progress over the last few years, there are still some opening questions regarding how to integrated new AI components. Moreover, multi-agent scenarios where the acting entities can perform low-level sensing and control tasks are becoming more and more available both in the industrial and academic environments. And, at the same time, epistemic planning is receiving a lot of interest.

In [31] and in [32] Capitanelli *et al.* propose a set of PDDL+ formulations that allows to model the problem of manipulating articulated objects in a three-dimensional workspace with a dual-arm robot. Instead, [7] address the same domains while encoding the planning problem directly in ASP making a strong use of macros. Another similar tool that inspired our research is the `ROSoClingo` [33] ROS package. This framework integrates Clingo [30] into the ROS service and `actionlib` architecture, providing an high-level ASP-based interface to control the behavior of a robot. While these works provided the foundation for our research and are far more complete tools, they do not consider neither the information flows between agents, nor their knowledge. This aspect is key in every multi-agent scenario, where reasoning on the perspective of others should be taken into consideration.

That is why we proposed `e-PICO`, a framework that offers the possibility of modeling multi-agent epistemic planning problems in robotics environments. This tool stems from the aforementioned approaches and integrates the latest MEP techniques to tackle the planning problems considering also the information flows. While integrating the epistemic aspect of multi-agent domain is, in our opinion, of the utmost importance, it requires high computational resources. That is why, `e-PICO` also introduces two methods to improve the planning times and to have feasible solving processes: **i)** a hierarchical usage of epistemic and classical planners to abstract and simplify the problems when considered by epistemic solvers; and **ii)** the employment of macros in epistemic planning.

Acknowledgments

The research presented in this paper is partially supported by Fondazione Friuli/Università di Udine project on *Artificial Intelligence for Human Robot Collaboration*, by INdAM-GNCS Projects NoRMA and CUP_E55F22000270001.

References

[1] M. Bhatt, E. Erdem, F. Heintz, M. Spranger, Cognitive robotics, 2016.
[2] S. Tonetta, ROBDT robotic digital twin, https://es.fbk.eu/index.php/projects/robdt/, 2021. Accessed: 2022-05-10.
[3] M. Bozzano, R. Bussola, M. Cristoforetti, M. Jonas, K. Kapellos, A. Micheli, D. Soldà, S. Tonetta, C. Tranoris, A. Valentini, Robdt: Ai-enhanced digital twins for space exploration robotic assets, in: The 2022 International Conference on Applied Intelligence and Informatics (AII2022), 2022.
[4] A. K. Jónsson, N. Muscettola, P. H. Morris, K. Rajan, Next generation remote agent planner, 1999.
[5] T. Niemueller, T. Hofmann, G. Lakemeyer, Goal reasoning in the clips executive for integrated planning and execution, 2019.
[6] T. Hofmann, T. Viehmann, M. Gomaa, D. Habering, T. Niemueller, G. Lakemeyer, C. Team, Multi-agent goal reasoning with the clips executive in the robocup logistics league., in: ICAART (1), 2021, pp. 80–91.
[7] R. Bertolucci, A. Capitanelli, C. Dodaro, N. Leone, M. Maratea, F. Mastrogiovanni, M. Vallati, An asp-based framework for the manipulation of articulated objects using dual-arm robots, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2019, pp. 32–44.
[8] M. Helmert, The fast downward planning system, Journal of Artificial Intelligence Research 26 (2006) 191–246.
[9] N. Lipovetzky, H. Geffner, Best-first width search: Exploration and exploitation in classical planning, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, California, USA, 2017, pp. 3590–3596. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14862.
[10] S. Richter, M. Westphal, The lama planner: Guiding cost-based anytime planning with

landmarks, Journal of Artificial Intelligence Research 39 (2010) 127–177. doi:10.1613/jair.2972.

[11] R. Fagin, J. Y. Halpern, Reasoning about knowledge and probability, Journal of the ACM (JACM) 41 (1994) 340–367. doi:10.1145/174652.174658.

[12] T. Bolander, A gentle introduction to epistemic planning: The del approach, arXiv preprint arXiv:1703.02192 (2017).

[13] J. Hintikka, Knowledge and Belief: An Introduction to the Logic of the Two Notions, Ithaca: Cornell University Press, 1962.

[14] C. Baral, G. Gelfond, E. Pontelli, T. C. Son, An action language for multi-agent domains: Foundations, arXiv preprint arXiv:1511.01960 (2015).

[15] H. Van Ditmarsch, W. van Der Hoek, B. Kooi, Dynamic epistemic logic, volume 337, Springer Science & Business Media, 2007.

[16] S. A. Kripke, Semantical considerations on modal logic, Acta Philosophica Fennica 16 (1963) 83–94.

[17] J. Gerbrandy, W. Groeneveld, Reasoning about information change, Journal of Logic, Language and Information 6 (1997) 147–169. doi:10.1023/A:1008222603071.

[18] F. Fabiano, A. Burigana, A. Dovier, E. Pontelli, EFP 2.0: A multi-agent epistemic solver with multiple e-state representations, in: Proceedings of the 30th International Conference on Automated Planning and Scheduling, 2020, pp. 101–109.

[19] F. Fabiano, A. Burigana, A. Dovier, E. Pontelli, T. C. Son, Multi-agent epistemic planning with inconsistent beliefs, trust and lies, in: D. N. Pham, T. Theeramunkong, G. Governatori, F. Liu (Eds.), PRICAI 2021: Trends in Artificial Intelligence - 18th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2021, Hanoi, Vietnam, November 8-12, 2021, Proceedings, Part I, volume 13031 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 586–597. URL: https://doi.org/10.1007/978-3-030-89188-6_44. doi:10.1007/978-3-030-89188-6_44.

[20] M. Gelfond, V. Lifschitz, Action languages, Electron. Trans. Artif. Intell. 2 (1998) 193–210. URL: http://www.ep.liu.se/ej/etai/1998/007/.

[21] V. Lifschitz, Answer set programming, Springer Berlin, 2019.

[22] A. Dovier, A. Formisano, E. Pontelli, Perspectives on logic-based approaches for reasoning about actions and change, in: M. Balduccini, T. C. Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, volume 6565 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 259–279. URL: https://doi.org/10.1007/978-3-642-20832-4_17. doi:10.1007/978-3-642-20832-4_17.

[23] A. Burigana, F. Fabiano, A. Dovier, E. Pontelli, Modelling multi-agent epistemic planning in ASP, Theory Pract. Log. Program. 20 (2020) 593–608. URL: https://doi.org/10.1017/S1471068420000289. doi:10.1017/S1471068420000289.

[24] Y.-q. Jiang, S.-q. Zhang, P. Khandelwal, P. Stone, Task planning in robotics: an empirical comparison of pddl-and asp-based systems, Frontiers of Information Technology & Electronic Engineering 20 (2019) 363–373.

[25] C. Baral, T. Bolander, H. van Ditmarsch, S. McIlrath, Epistemic Planning (Dagstuhl Seminar 17231), Dagstuhl Reports 7 (2017) 1–47. URL: http://drops.dagstuhl.de/opus/volltexte/2017/8285. doi:10.4230/DagRep.7.6.1.

[26] F. Fabiano, B. Srivastava, J. Lenchner, L. Horesh, F. Rossi, M. B. Ganapini, E-PDDL: A standardized way of defining epistemic planning problems, CoRR abs/2107.08739 (2021). URL: https://arxiv.org/abs/2107.08739. arXiv:2107.08739.

[27] A. H. C. Robert Haschke, Chris Lalancette, Rviz documentation, http://wiki.ros.org/rviz/UserGuide, ???? Accessed: 2022-02-24.

[28] D. McDermott, M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, D. Weld, PDDL - The Planning Domain Definition Language, Technical Report, Technical Report, 1998.

[29] M. Fox, D. Long, Pddl2.1: An extension to pddl for expressing temporal planning domains, Journal of Artificial Intelligence Research 20 (2003) 61–124. URL: http://dx.doi.org/10.1613/jair.1129. doi:10.1613/jair.1129.

[30] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, CoRR abs/1705.09811 (2017).

[31] A. Capitanelli, M. Maratea, F. Mastrogiovanni, M. Vallati, Automated planning techniques for robot manipulation tasks involving articulated objects, in: Conference of the Italian Association for Artificial Intelligence, Springer, 2017, pp. 483–497.

[32] A. Capitanelli, M. Maratea, F. Mastrogiovanni, M. Vallati, On the manipulation of articulated objects in human–robot cooperation scenarios, Robotics and Autonomous Systems 109 (2018) 139–155.

[33] B. Andres, P. Obermeier, O. Sabuncu, T. Schaub, D. Rajaratnam, Rosoclingo: A ros package for asp-based robot control, arXiv preprint arXiv:1307.7398 (2013).

CARING for xAI

Flavio Bertini[1,†], Alessandro Dal Palù[1,*,†], Francesco Fabiano[1,†] and Eleonora Iotti[1,†]

[1]*Department of Mathematical, Physical and Computer Sciences, University of Parma, Parco Area delle Scienze 53/A, 43124, Parma, Italy*

Abstract

Over the last few years, *Artificial Intelligence* (AI) has pervaded our lives. As a result, automated tools that "reason" on different scenarios have become more and more common. As this trend continues to grow, it has become necessary to ensure that newly developed tools and technologies can be safely adopted, as demonstrated by the numerous EU regulations. This is especially true when the concept of AI is intertwined with the field of medicine, where every decision may be critical. That is why, in this work, we decided to tackle the problem of *automated interpretation* of *Computed Tomography* (CT) *scans* using an *explainable* approach. In fact, while several methods based on Machine Learning (ML) are currently available, these are still outperformed by medical doctors and provide answers that cannot be traced back to a logical deduction. This paper presents CARING, a new methodology based on Answer Set Programming (ASP), which returns reliable, easy-to-program and explainable interpretations of CT scans. In particular, CARING makes use of transparent technologies in order to handle medical knowledge provided either by experts or by verified ontologies. This proof of concept shows that Logic Programming is a mature technology that can match the newest challenges in the xAI field.

Keywords

Explainable AI, Answer Set Programming, CT Scan, Image processing, Tissue Segmentation

1. Motivation

This paper investigates the design of CARING (**C**T-scan **A**utomated **R**easoning **IN**terpretation **G**uidance)[1]. The purpose of this tool is to implement an AI pipeline that reproduces the mental process of analysis and interpretation of a CT scan performed by radiologists. The input of the tool is the set of raw data from the acquisition system and the output is the labelling of organs and structures contained in the body scan. The goal of the tool is to provide the basis for further automated analyses and to support Medical Doctor decisions.

The problem contains various challenges that combine artificial vision and reasoning tasks. Even if the problem has been largely studied in the literature, the attempts proposed so far (see Section 2.1) can be classified as black box approaches: the user has little control about accuracy and accountability of system's results; introducing/updating medical information to the system

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

*Corresponding author.

†These authors contributed equally.

flavio.bertini@unipr.it (F. Bertini); alessandro.dalpalu@unipr.it (A. D. Palù); francesco.fabiano@unipr.it (F. Fabiano); eleonora.iotti@unipr.it (E. Iotti)

0000-0001-6925-5712 (F. Bertini); 0000-0003-0353-158X (A. D. Palù); 0000-0002-1161-0336 (F. Fabiano); 0000-0001-7670-2226 (E. Iotti)

[1]ahead-lab.unipr.it/caring

require significant rewriting/training. According to recent EU regulations about *explainable AI* (see Section 2.2), Medical Doctors can include an AI based system in their workflow under certain requirements. The most relevant ones are that the system must provide a support or proof for its outputs. We believe that this perspective poses interesting challenges to the Logic Programming (LP) community, since this paradigm naturally supports such requirements.

Radiologists, when interpreting the content of a CT scan, typically start from spatially local details, find relations among regions and eventually recognize all anatomic details. Local details are combined into geometric regions and (higher-level) medical knowledge allows to map relations among local details, in order to support the proposed classification. The process has to cope with a remarkable body's variability: *e.g.*, organs can appear in different sizes, locations and shapes. Structural relationships are by far more preserved, even if several anatomical structural variants are documented (see, *e.g.*, [1] for blood vessel ones).

In this context, details in the scans, or *features*, are represented by visible surfaces that separate regions with different properties (hypothetical boundary between organs/tissues) and uniform regions that may belong to the same object. These features can be easily noticed in the scan, even by an untrained viewer, and they are one of the first points of attention in the analysis performed by radiologists. Each feature does not necessarily correspond to a specific tissue or organ, but this first step of abstraction has the effect of reducing the computational burden of the next phase. Common edge detection and segmentation algorithms (classical off-the-shelf tools, Machine Learning approaches, Neural Networks) often go beyond simple geometric analyses and encode some higher level strategy to resolve some ambiguities to deliver a more accurate result. Unfortunately, such approaches hard code some declarative properties about the domain that can not be easily controlled as high level properties (*e.g.*, there are parameters to be tuned; a trained network provides no understanding of the rules that drive the process). We find here two main issues: (i) the algorithms' output provides no high level explanation of the assumptions made, (ii) it is difficult and/or impossible to modify the behaviour of such algorithms depending on some high level requirements.

Such scenario suggests a processing pipeline where the identification of pure geometric basic elements is decoupled from reasoning about their possible anatomic role. In order to create an explainable reasoning, geometric feature extraction must operate with the lack of any medical information.

The goal is to delay the interpretation process to the reasoning phase. The advantage is that the reasoning is completely in charge of exploiting medical knowledge and it is able to give proof of every hypothesis and deduction process.

The paper presents a background Section 2 with a brief overview on CT scans, explainable AI, medical ontology and LP methods employed. Section 3 introduces a CSP based geometric feature extraction and Section 2.3 describes the ASP based reasoner on geometric features. In Section 5 we show some preliminary results and in Section 6 we conclude.

Let us conclude the introduction with a summary of the advantages of having an LP core inside the pipeline:

- medical knowledge is modelled in a LP framework. It can be handled as a collaborative knowledge base (with possibility to merge inconsistent facts; retrieve legal liability of authors of input knowledge; incremental and distributed management) and/or as ontology;

- medical knowledge can be translated into and from natural language for easier interaction with Medical Doctors;
- the reasoner offers a transparent approach and the output is explainable: the set of activated rules that support the results can be provided, discussed and argued by medical doctors, in the same way colleague's opinions are compared and this can stimulate knowledge base reviewing, in a constant process of refinement;
- lack of information, incoherent knowledge, uncertainty and hierarchical labeling can be handled;
- there is no need for training sets (like in neural network cases). Knowledge base can be built either manually, but this task is less time consuming, compared to setting up training sets (hundreds of CT scans with manual classification of volumes), or automatically from ontologies processing.

2. Background

2.1. CT scans

Computer Assisted Tomography (in short CT) scan [2] is a well established X-ray based acquisition procedure suitable for diagnoses of pathology and traumas as well as surgical preparation procedures. A patient's body image is acquired and rendered as a 3D volumetric information about X-ray absorption performed by various objects composing human body. Each *voxel* (volume element) of the scan has a typical size of 1 mm^3. Such volumetric information is arranged in a 3D matrix, where each cell contains a scalar number which corresponds to the radiometric absorption of that specific voxel (from now on *intensity*). The unit of measure is the HU (Hounsfield Unit). Different chemical elements, as well as different body structures, cause different X-ray absorptions. Therefore, 3D images can be used to identify the location of organs, vessels, bones, etc. and to assess their correct functional status. Note that different structures can show similar intensity levels. At the same time acquisition noise can reduce the clear perception of smaller issues. The goal of the training of radiologists is to cope with such issues and to be able to recognise finer details and problems. For example, typical tumours are less dense than healthy tissues and therefore they absorb less X-rays. This fact can be visualized on a 3D image as a darker area compared to the neighborhood, assuming to depict values with shades of gray ranging from black (low absorption, *e.g.*, air) to white (high absorption, *e.g.*, bones). However, while small tumours are more difficult to be identified, the prognosis is more favourable in case of early detection. Therefore computer assisted analysis have gained attention over the last decade, in order to improve the detection rate. There are various domains of analysis for a CT scan: *e.g.*, vision related ones are about organ and multi-organ segmentation (see [3] for a survey) and cancer detection (*e.g.*, [4]). Interestingly most methodologies, despite their sensible precision, are based on methods that are not xAI compliant (see Section 2.2). Our goal is therefore to design a novel pipeline that does not rely on such methods.

2.2. Explainable Artificial Intelligence

In the last decades we witnessed a very fast growing advances in AI based systems. In particular, from 2012 onward, sub-symbolic techniques such as Deep Learning (DL) became de-facto standards to deal with Computer Vision challenges, especially classification tasks such as recognition. Furthermore, they also showed great and increasing performances in object detection and image segmentation [5]. Another example of success is in natural language processing challenges with remarkable results in text generation [6]. This advance is also due to the accessibility of huge amount of information used to train systems, and by the availability of increasingly powerful hardware architectures that made it possible to use optimization algorithms on very large parameter spaces, such as those of deep neural networks. The measure of success of such techniques can be observed in how much AI is becoming more and more pervasive in daily life. Sub-symbolic methods heavily rely on learning algorithms over latent spaces that consist of billions of parameters and (in case of neural networks) hundreds of hidden layers, whose final goal is to learn approximations of an answer. Therefore, beside the impressing results obtained by DL, such approximations have been subject to criticism about their weakness to biasedness and the subsequent lack of robustness [7]. As another significant drawback of DL-based pipelines, is that such parametric spaces have to be treated as black-boxes, working as provider of intuitive information rather than transparent and reasoning-based decisions. For most situations the lack of transparency is not a problem, but their application to critical areas becomes an issue. In AI systems supporting medical decisions, the ability of interpret and explain the recommendations provided by the system is crucial. This also has to do with the concept of trustworthiness and reliability with reference to an *audience* of human users [8]. In recent years, the focus on ethical challenges in AI leads to move towards the so called *eXplainable Artificial Intelligence* (xAI), which aims at providing transparent and interpretable models and thus to increase trust in system based decisions. Explainability, and thus xAI, finds different definitions in literature, according to two main different approaches, namely the integration of inherently interpretable techniques in the system design, and the post-hoc explanation of what the system already have been done. The latter method is mostly employed in DL based AI, providing later description of the decision rather than unwrapping the black-box [9]. Such a method is yet under discussion, and concepts of explainability and causability in DL-based approaches are far from being attained.

In this work, the need of an intrinsically interpretable system is motivated by the degree of criticality of the task. As a matter of fact, it is desirable to introduce xAI techniques where the central point of the system regards the health of a patient and the detection of system errors of must be early and easy. The transparency allows, in fact, the users (Medical Doctors) to make informed choice and to be more confident when using the program. For this and other reasons, the topic of xAI in critical systems is being addressed by the European Commission, which developed an AI strategy to rule the trustworthiness of AI systems and enhance the excellence in such field [10, 11]. The development of such a strategy which led to a complex legal framework is motivated by the risk factors in the employment of AI-based systems in everyday life, as they became more and more pervasive as already said. In particular, one of the main reasons provided by the EU Commission is that whenever is not possible (or difficult) to find out why a decision or prediction has been made by an AI system, it consequently becomes not possible (or difficult) to detect whether someone has been unfairly disadvantaged. An AI improvement, according to the EU legal framework, must be subject to rules in order to protect the functioning of markets and the public sector, people's safety and fundamental rights. Such a legal framework divides AI systems in a pyramidal structure that identifies four degrees of risk, namely, minimal risk, limited risk, high risk and unacceptable risk. While unacceptable risk AI systems are clearly recognized as a threat for people's safety, and thus banned or prohibited, high risk systems comprises many AI technologies used in critical settings, such as health. Limited risk refers to systems which need specific transparency duties, such as chatbots, and finally, minimal risk are the vast majority of common systems, covering all applications that make everyday tasks more comfortable and/or easier. For the scope of this work, the proposed AI system falls in the high risk and thus subject to specific obligations before it can be proposed as a complete working system. Those obligations comprise (but are not limited to) the use of high quality of datasets as input of the system, in order to minimize discriminatory or riskful outcomes, the traceability of results, and the providing of clear and adequate information to the user. In particular, "High-risk AI systems shall be designed and developed in such a way to ensure that their operation is sufficiently transparent to enable users to interpret the system's output and use it appropriately." [11].

2.3. Human Anatomy Ontology

Clinical medicine uses standards for many types of data ranging from diseases (*International Classification of Diseases*[2] - ICD) to diagnoses and procedures (*Systematized Nomenclature of Medicine*[3] - SNOMED) and from laboratory data (*Logical Observation Identifiers Names and Codes*[4] - LOINC) to imaging data (*Digital Imaging and Communications in Medicine*[5] - DICOM). One of the earliest hierarchical terminology developed allows for cataloguing biomedical information and indexing journal articles (*Medical Subject Headings*[6] - MeSH). This controlled vocabulary is still used for MEDLINE, PubMed and National Library of Medicine databases. However, there are more formal and rich mechanisms to represent concepts and relationships, that is ontologies. In Computer Science, an ontology is a mean to formally model information and knowledge in a specific domain using simple representational primitives, such as classes, attributes, and relations among the members of the classes [12]. The concepts and relations belonging to the ontology are typically specified in a language that abstracts implementation strategies. In the literature, these languages are classified in various ways, however, they are commonly based on either first-order logic or description logic [13]. In particular, the Web Ontology Language (OWL) is a kind of language endorsed by the World Wide Web Consortium and conceived when the information needs to be processed by applications instead of just presented to humans [14]. In particular, OWL features include a collection of expressive operators for the concept, properties and relationships description, the ability to specify characteristics of properties (*e.g.*, transitivity, domains and ranges), and a semantic facilitating the use of inference and

[2]ICD: https://icd.who.int.
[3]SNOMED: https://www.snomed.org.
[4]LOINC: https://loinc.org.
[5]DICOM: https://www.dicomstandard.org.
[6]MeSH: https://www.nlm.nih.gov/mesh/meshhome.html.

reasoning [15]. The use of computational reasoning to answer complex questions has increased the development of ontologies in several medical fields supporting precision medicine and trustworthy artificial intelligence [16]. In biomedical and health sciences, ontologies are used to represent knowledge in different areas. The Disease Ontology extensively uses medical terminology standards (*e.g.*, MeSH, ICD, SNOMED) to semantically integrates disease and medical vocabularies [17], whereas the Infectious Disease Ontology is a set of interoperable ontologies covering the clinical aspects and the pathogens of most infectious diseases [18]. Other ontologies address the need for consistent descriptions of gene products across databases [19], represent phenotypic abnormalities encountered in human disease [20], annotate clinical trials [21], and support automated reasoning based on vaccine knowledge [22]. The Foundational Model of Anatomy ontology (FMA) represents a declarative knowledge about the whole human anatomy [23]. The last version of FMA (ver. 5.0.0) consists of 104,721 classes, organised hierarchically, and 168 properties that describe various characteristics of the concepts, such as "`constitutional_part_of`", "`adjacent_to`", "`dimension`" and "`bounds`", to name a few. For instance, the *Left lung* (code FMA:7310) is "`subclass_of`" *Lung* (code FMA:7195); is a "`constitutional_part_of`" *Left pulmopleural compartment* (code FMA:85056), *Left hemithorax* (code: FMA:20360) and *Intrathoracic part of chest* (code FMA:73438); and is related as "`regional_part_of`" to the *Lower lobe of left lung* (code FMA:7371) and *Upper lobe of left lung* (code FMA:7370). In this work, FMA codes of some high-level concepts are used to create the labels to be assigned to the cluster of voxels.

2.4. Constraint Satisfaction Problem

The field of automated reasoning requires ways of formalizing and describing scenarios and procedures with a great level of abstraction [24]. From this need stemmed the well-known *logic/declarative programming* paradigms. In contrast with the more common *imperative* and *object-oriented* paradigms, that require precise lists of instructions, the declarative ones "simply" need to state constraints and objective function [25]. This allows to describe a problem, and its solution, without the distraction of algorithmic details. Among various declarative paradigms, one of the most mature and employed in AI is *Constraint* Programming (CP) that, as said by [26], "is a powerful paradigm for solving combinatorial search problems". CP directly derives from the *constraint satisfaction problem*, that is the process of finding a solution through a set of constraints that impose conditions that the variables must satisfy [27]. Several techniques, that range from *search*, *constraint propagation*, and *backtracking*, are employed in solving CP models. Due to space limitations we will not provide further details on the topic and we address the interests readers to [26, 27] for a complete introduction on the topic of constraint programming.

In our work we utilize CP to identify sub-partitions of CT scans' regions (see Section 3.2). The problems that need to be solved are dynamically generated with constraints that depend on the analyzed regions. This can be easily modeled into constraint satisfaction problems, as we know the set of rules and constraints that a set of voxels must respect to be considered as a sub-region. Instead, trying to generate an imperative procedure flexible enough to tackle all the various cases could result impossible given the great variety of cases that different regions generate.

2.5. Answer Set Programming

Answer Set Programming (ASP) is a language that derives from the logic programming paradigm and that stems from the idea of *stable models* [28]. In particular, ASP can be used to model diverse domains through values, facts, variables, rules, and constraints. These domains are then solved trying to find a possible set of variables assignments, *i.e.*, a variable, if considered by the solution, must be assigned to a value without contradicting any constraint. More formally, as said by [29], a program P in the language ASP is formed by set of rules r of the form:

$$a_0 \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n$$

where $0 \leq m \leq n$ and each element a_i, with $0 \leq i \leq n$, is an *atom* of the form $p(t_1, \ldots, t_k)$, p is a predicate symbol of arity k and $t_1, \ldots, t_k$ are terms built using variables, constants and function symbols. Negation-as-failure (naf) literals are of the form $not\ a$, where a is an atom. Let r be a rule, we denote with $h(r) = a_0$ its *head*, and $B^+(r) = \{a_1, \ldots, a_m\}$ and $B^-(r) = \{a_{m+1}, \ldots, a_n\}$ the positive and negative parts of its *body*, respectively; we denote the body with $B(r) = \{a_1, \ldots, not\ a_n\}$. A rule is called a *fact* whenever $B(r) = \emptyset$; a rule is a *constraint* when its head is empty ($h(r) = \mathsf{false}$); if $m = n$ the rule is a *definite rule*. A *definite program* consists of only definite rules.

A term, atom, rule, or program is said to be *ground* if it does not contain variables. Given a program P, its *ground instance* is the set of all ground rules obtained by substituting all variables in each rule with ground terms. In what follows we assume atoms, rules and programs to be *grounded*. Let M be a set of ground atoms ($\mathsf{false} \notin M$) and let r be a rule: we say that $M \models r$ if $B^+(r) \nsubseteq M$ or $B^-(r) \cap M \neq \emptyset$ or $h(r) \in M$. M is a *model* of P if $M \models r$ for each $r \in P$. The *reduct* of a program P w.r.t. M, denoted by P^M, is the definite program obtained from P as follows: (i) for each $a \in M$, delete all the rules r such that $a \in B^-(r)$, and (ii) remove all naf-literals in the the remaining rules. A set of atoms M is an *answer set* [28] of a program P if M is the minimal model of P^M. A program P is *consistent* if it admits an answer set.

The declarative nature of ASP allowed us to define a solid base containing the anatomical knowledge derived by ontologies or medical operators. We then used this knowledge base in our tool to autonomously classify the CT scan respecting the information provided by the experts. A more detailed explanation of how this process was envisioned and realized will be presented in Section 4.1.

3. Features extraction

3.1. Pre-processing and smoothing

One relevant issue about CT scan is that acquisition noise can be significant and it can even mask tumors. Radiologists are trained to mentally filter out noise and investigate scan content. Their first task is to correctly recognize the objects in the scan so they can then observe finer details, where texture (typical local arrangements of intensity absorption) can suggest potential pathologies. Often texture and noise merge and make the process difficult.

In this paper we focus on the first task and we set the general goal to interpret any voxel of the scan. This goes beyond the common goal in the literature where multi-organ detection targets main organs (liver, spleen, pancreas etc). Our approach could potentially identify any visible object according to a general ontology. Single organ detection precision, reached through DL, is still rather unsatisfactory, *e.g.*, up to 86% of pancreas volume is correctly classified [30]. We believe that the classification problem should be solved with a holistic approach, *i.e.*, an evaluation that considers at the same time every property about every element.

The processing pipeline starts with a rather strong smoothing of data. The resulting scan has little to no texture but it preserves boundaries between different regions. Organs and tissues are made of similar intensity voxels and any surface that separates such voxels is a suggestion to be considered during reasoning.

We selected a Total Variation based 3D smoothing named Directional Fast Gradient Projection [31]: this family of filters is particularly effective in CT scan domains and in surface preserving even in presence of strong noise. In Figure 1 we show in a rainbow palette the comparison between original raw data and smoothed version for a slice of a scan that intersects two vertebrae (red-blue color), a part of the kidney (top right dark green) and skin of the back (red bottom line). Cyan color is air. It can be noticed how structures are well preserved, while texture and noise are almost flattened.

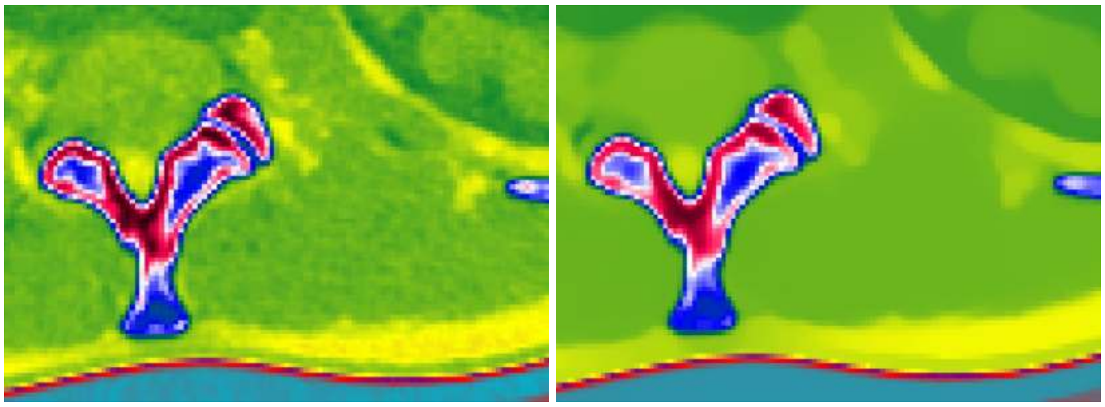


Figure 1: Raw data (left) and smoothed data (right)

The filtered version is processed with the goal of retrieving regular 3D regions. Unfortunately simple thresholding can not be applied, even if typical HU values are well documented: same intensity values can belong to different organs.

3.2. Geometric clustering

We set up a detection of regions that are defined in terms of simple mathematical relations. The resulting regions have little relation to target organs, but they can be handled as equivalence class for clustered voxels.

The idea is to group together voxels that have similar intensity and that are separated by sudden changes (high gradients). We show in Figure 2 on the left a simplification on a 2D domain. The same concepts can be generalized in the 3D space. Each point, associated to a voxel and a local intensity value, has a local neighbor (in the 3x3x3 window) with highest intensity (except for the local maximum). A graph induced by this relation defines connected components that reach a local maximum (dark blue and red points). This already represents a partitioning of the space, with the property that only monotonic paths are contained. A typical smooth CT scan, made of 512^3 voxels may host some million local maxima. Inside a region, a finer analysis can reveal that along path there are maximal gradients values that cooperate to form a cut surface (see dashed line and black arrows that separate green and light blue points). It is important to separate such sub-partitions, since they represent visible changes that suggest an interface between different objects. In Figure 2 on the center, we depict with different colors the 3D points belonging to the same region but to different sub-regions: it can be seen that the bone-muscle interface (white - blue - green voxels, only one scan slice is depicted) is correctly partitioned by green and blue points (each point represents voxels in neighbor planes). Moreover the set of red points correctly suggests another division between white and blue voxels (different inner structures of the bone).

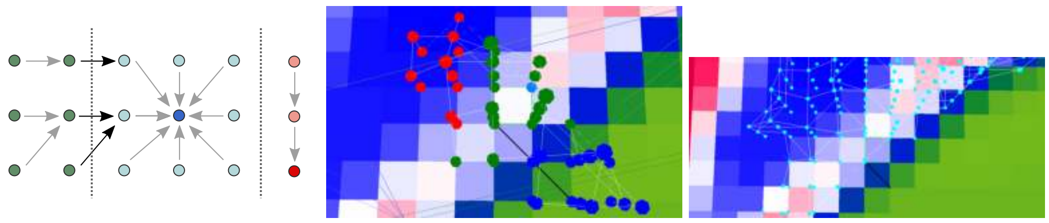


Figure 2: Example of geometric clustering. (left) Maximal gradient approximation for regions, (center) best surface cuts and sub-partitions, (right) clustering of sub-partitions.

In order to identify such sub-partitions inside a region, we set up a Minizinc [32] model that optimizes the best partitions that provide the highest cut in the corresponding graph. Each point is a variable that can be assigned to a partition code (3 different codes are enough for most cases). Each edge, defined above, is associated to the gradient magnitude. The constraints impose that the partition codes must satisfy the precedence imposed by edges directions. The only other constraint is that the local maximum is assigned to the maximum code (*i.e.*, 2 for our tests). The cost function is the sum of the edges that connect two different codes points. We found that Coin BC solver [33] was the most efficient among the ones provided by default Minizinc interface.

The resulting sub-partitions are almost constant valued, but their number is still very large to be used for reasoning. We also introduce a clustering that merges neighboring sub-partitions that also share same intensity and/or gradient directions along the shared surface of contact. Again, the resulting clustered voxels have similar intensities, but they collect a large set of sub-partitions. In Figure 2 on the right, we show with cyan points how the green sub-partition in the central picture gets merged with compatible neighboring partitions (in this case all red-white colored voxels).

The result of this procedure is a manageable set of (several thousands of) clusters that are retrieved by no medical information exploitation.

4. Ontology/Medical Knowledge Mapping into the Graph

Geometric clustering provides a set of partitioned volumes with annotated features that are considered from now on as the nodes of our search graph. Edges of such graph connect two volumes that share some surface of contact.

The goal of the labeling is to assign to each node a FMA code from the ontology. Since anatomic ontology is naturally hierarchical, we adapt our graph coloring like procedure in order to account for a hierarchical handling of FMA codes for domains. Ideally the root of the ontology represents any possible body's object, while following a particular branch in the ontology, it specializes the assignment for the specific node. Our optimal solution assigns the lowest label in the ontology to each node. Most quality results gets leaf values from the ontology.

Medical knowledge determines the constraints over ontology values and specify how spatial relations, intensity values and anatomic relations among objects apply.

We therefore define a set of ASP rules (the medical knowledge) and an ASP program that receives as input the graph structure as processed by the feature extraction.

4.1. ASP for Models Generation

As already mentioned, one of the main objectives of this work is to provide a tool that is able to support medical experts while providing proofs for its outputs. To accomplish this we decided to employ Answer Set Programming, through the Clingo solver [34], to analyze the information, expressed in the form of a graph, extrapolated by the scans. The motivation behind this choice is threefold: *i)* thanks to its declarative programming, ASP allows us to express anatomical rules in a concise and effective manner; *ii)* the process that leads ASP in finding the best model is completely traceable and only derived by the rules that reflect actual medical knowledge; and *iii)* finally, given the declarative nature of ASP and the consequent succinctness, enriching the program with new medical knowledge can be done fairly easily.

The main role that the ASP module of CARING undertakes is to identify the right class, w.r.t. the standard classifications introduced in Section 2.3, in which each node of graph falls into. This is done by inheriting the hierarchical approach typical of the medical knowledge.

In particular, we envisioned our ASP model to "firstly" identify the macro categories (the first level of FMA) of the various nodes. This means that each node is assigned to one, or more, of the following classes: `air`, `lung`, `fat`, `tissue`, `bone` or `unknown`. To do so we define the possible labels that each node can be assigned to by looking at their `intensity` values. While most of the nodes falls perfectly into only one category, some of them have values that can belong to multiple labels or even to none (that is why we also introduced the `unknown` category). The possible labels that a node `N` can be assigned to are captured by the set of ASP predicates `poss_lab(N,L)`, where `N` is the node id and `L` is one of the aforementioned labels. This predicate is defined as follows:

$$\begin{aligned}\mathtt{poss\_lab(N,L)\!:\!-}\quad & \mathtt{node(N,I), lab(L), in\_min(L,MIN), in\_max(L,MAX),}\\ & \mathtt{in\_tolerance(L,T), I + T \geq MIN, I - T < MAX.}\end{aligned}$$

where: *i)* `node(N,I)` is the description of a node, in terms of id and intensity, provided by the features extraction procedure; *ii)* `lab(L)` express that `L` must belong to the aforementioned set of labels; *iii)* `in_min(L, MIN)` and `in_max(L, MAX)` describe the range in which a label `L` is defined by providing its minimum and maximum intensity, respectively; and *iv)* `in_tolerance(L,T)` provides the tolerance `T` of the intensity range for the label `L`.

After generating the possible label domain for each node we try to minimize the number of unknown. This is done by allowing the unknown nodes to assume their neighbors' labels if their intensity is not too distant. The ASP model will return, as solution, the one that minimize the number of unknown nodes. An example of one of the rules used to reduce the unknown is as follows:

```
poss_lab(N,L):-  poss_lab(N,unknown),node(N,I),lab(N2,L),L! = unknown,
                 edge(N,N2,_,_),in_tolerance(L,T1),T = T1 * 2,
                 in_min(L,MIN),in_max(L,MAX),I + T ≥ MIN,I − T ≤ MAX.
```

where: *i)* `lab(N2,L)` is the label assigned to `N2` in the solution of the model, generated by "`1{lab(N,L) : poss_lab(N,L)}1:-node(N,_).`"; and *ii)* `edge(N,N2,_,_)` is the fact, derived by the imperative procedure, that indicates that nodes `N` and `N2` are connected in the graph.

Finally, the ASP program also discriminates among different entities that fall in the same category. In particular, for this work, we focused on distinguishing the various bones that were present in our working area. To do so we defined some rules that relate intensity of the nodes, labelled as bones, in relation to the minimal and maximal intensity of the contact surface with other bone-labelled nodes. This allowed us to identify bone-labelled nodes with a strong difference between the average intensity and the intensity on the contact surface, hinting that the two connected nodes may belong to two different bones. This is because two bones that are in contact are always separated by lower intensity zones. We then impose that the nodes that belong to the same bone must be transitively connected. Some results of this classification process are presented in Figure 3.

This is a minimal test-bed domain that we used to show our the resolution schema rather than the overall performances. The accuracy of the program, in fact, is under constant improvement as it is extended to cover additional regions and sub-labels.

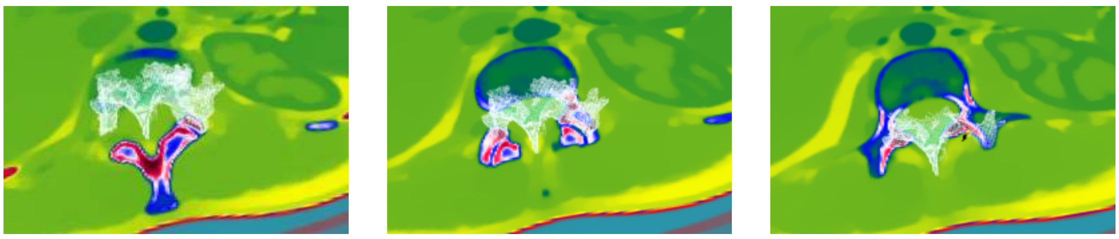


Figure 3: An example of result of the CARING procedure. The tool allows to extract the set of voxels that describe specific anatomical objects. In this case we rendered (with a white surface) the set of point that CARING assigned to a single vertebra. From left to right we see the vertebra rendered on different slices of the scan.

Let us note that, even if our description of the ASP model is presented as a sequence of properties, the solving process of ASP takes advantage of all conditions simultaneously. We refer the interested reader to [35] for an insightful explanation of this solving process.

5. Results

Our proof of concept processed a raw abdominal CT scan with size 512x512x70 voxels. Total variation filtering on a P100 Nvidia GPU took a few seconds of processing. We focused on a region with 110 x 110 x 38 voxels that covers two vertebrae, one rib, the lower end of a lung, part of a kidney and spleen.

Geometric clustering for the whole scan produced 740K maximal voxels (average of 25 voxels per partition), while the region of interest contained 5K maximal voxels (average of 83 voxels per partition). Computing sub-partitions with Minizinc (5K CSPs solved in a batch task) took 80 minutes on a 2.3GHz i9 Macbook Pro. This phase will require some optimization/reencoding in order to make it scalable for complete scans (100x–1000x of speedup needed).

Minizinc computation returned 22K sub-partitions, that were clustered into 5355 clusters. Based on intensity ranges, we directly labelled 3931 clusters with most general FMA code. The ASP procedure labelled 1424 clusters with a time of 0.602 seconds. The procedure is still in its infancy, as the accumulated medical knowledge is far from being complete. Nonetheless CARING is able to discriminate between the macro categories of anatomical bodies and to label geometric features into compatible clusters. Our proof of concept implements the complete xAI pipeline and shows the feasibility of the approach. Our current knowledge base is limited but we expect promising results with a more extended set of properties.

6. Conclusions

In this paper we described the design of an xAI pipeline for the interpretation of CT scans. We covered the foundational aspects that helped us to guarantee explainability of the deduction process. As proof of concept, we implemented a first model that was able to analyze a region of a CT scan and to recover correct classification for contained regions.

This work can be extended along several directions: *i)* quality and efficiency improvement for the feature extraction phase; *ii)* increase the scalability of the ASP reasoner to tackle complete CT scans; *iii)* a multi level and hierarchical search for labeling ontology codes; *iv)* the definition of a knowledge base automatically extracted from FMA codes and their relationships; *v)* the possibility to manually add knowledge, with handling of liability and handling of inconsistent knowledge; *vi)* generate natural language descriptions for Medical Doctors' validation; *vii)* testing over challenging scans; and *viii)* comparison with the state of the art non-xAI systems.

Acknowledgments

We would like to thank Medical Doctors Mirko D'Onofrio and Nicolò Cardobi (University of Verona) for providing CT scan data and useful discussions.

References

[1] F. Wacker, H. Lippert, R. Pabst, Arterial variations in humans: Key reference for radiologists and surgeons, Thieme (2017).

[2] G. T. Herman, Fundamentals of computerized tomography: image reconstruction from projections, Springer Science & Business Media, 2009.

[3] M. Kardell, Automatic segmentation of tissues in ct images of the pelvic region. master thesis, 2014.

[4] L. C. Chu, et al., Application of deep learning to pancreatic cancer detection: Lessons learned from our initial experience, Journal of the American College of Radiology 16 (2019) 1338–1342.

[5] A. Krizhevsky, et al., Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012).

[6] T. Brown, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.

[7] G. Marcus, Deep learning: A critical appraisal, arXiv preprint arXiv:1801.00631 (2018).

[8] A. B. Arrieta, et al., Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai, Inf. fusion 58 (2020) 82–115.

[9] A. Holzinger, et al., Causability and explainability of artificial intelligence in medicine, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 9 (2019).

[10] E. Commission, A european approach to artificial intelligence, https://digital-strategy.ec.europa.eu/en/policies/european-approach-artificial-intelligence, 2022.

[11] E. Commission, Proposal for a regulation of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts, https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52021PC0206&from=EN, 2021.

[12] N. Guarino, D. Oberle, S. Staab, What is an ontology?, in: Handbook on ontologies, Springer, 2009, pp. 1–17.

[13] J. Pulido, M. Ruiz, R. Herrera, E. Cabello, S. Legrand, D. Elliman, Ontology languages for the semantic web: A never completely updated review, Knowledge-Based Systems 19 (2006) 489–497.

[14] D. L. McGuinness, F. Van Harmelen, et al., Owl web ontology language overview, W3C recommendation 10 (2004) 2004.

[15] S. Bechhofer, M. T. Özsu, L. Liu, Owl: Web ontology language, in: {Encyclopedia of Database Systems}, Springer Nature, 2009.

[16] M. A. Haendel, C. G. Chute, P. N. Robinson, Classification, ontology, and precision medicine, New England Journal of Medicine 379 (2018) 1452–1462.

[17] L. M. Schriml, et al., Human disease ontology 2018 update: classification, content and workflow expansion, Nucleic acids research 47 (2019) D955–D962.

[18] A. Ruttenberg, A. Goldfain, A. Diehl, B. Smith, B. Peters, Infectious disease ontology, The National Center for Biomedical Ontology 5 (2016).

[19] M. Ashburner, et al., Gene ontology: tool for the unification of biology, Nature genetics 25 (2000) 25–29.

[20] S. Köhler, et al., The human phenotype ontology in 2021, Nucleic acids research 49 (2021)

D1207–D1217.

[21] B. Smith, et al., The obo foundry: coordinated evolution of ontologies to support biomedical data integration, Nature biotechnology 25 (2007) 1251–1255.

[22] Y. Lin, Y. He, Ontology representation and analysis of vaccine formulation and administration and their effects on vaccine immune responses, Journal of biomedical semantics 3 (2012) 1–15.

[23] N. F. Noy, M. A. Musen, J. L. Mejino Jr, C. Rosse, Pushing the envelope: challenges in a frame-based representation of human anatomy, Data & Knowledge Engineering 48 (2004) 335–359.

[24] O. Torgersson, A note on declarative programming paradigms and the future of definitional programming, in: In Proceedings of Das Wintermote 96, 1996.

[25] A. Bockmayr, J. N. Hooker, Constraint programming, in: K. Aardal, G. Nemhauser, R. Weismantel (Eds.), Discrete Optimization, volume 12 of *Handbooks in Operations Research and Management Science*, Elsevier, 2005, pp. 559–600. URL: https://www.sciencedirect.com/science/article/pii/S0927050705120106. doi:https://doi.org/10.1016/S0927-0507(05)12010-6.

[26] F. Rossi, P. van Beek, T. Walsh, Handbook of Constraint Programming, Elsevier Science Inc., USA, 2006.

[27] E. P. K. Tsang, Foundations of constraint satisfaction., Computation in cognitive science, Academic Press, 1993.

[28] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming., in: ICLP/SLP, volume 88, 1988, pp. 1070–1080.

[29] F. Fabiano, A. Dal Palù, An ASP approach for arteries classification in CT scans, J. Log. Comput. 32 (2022) 331–346.

[30] Y. Zhou, L. Xie, W. Shen, E. Fishman, A. Yuille, Pancreas segmentation in abdominal ct scan: a coarse-to-fine approach, arXiv preprint arXiv:1612.08230 (2016).

[31] M. J. Ehrhardt, M. M. Betcke, Multicontrast mri reconstruction with structure-guided total variation, SIAM Journal on Imaging Sciences 9 (2016) 1084–1106.

[32] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack, Minizinc: Towards a standard cp modelling language, in: C. Bessière (Ed.), Principles and Practice of Constraint Programming – CP 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 529–543.

[33] J. Forrest, et al., coin-or/cbc: Release releases/2.10.8, 2022. URL: https://doi.org/10.5281/zenodo.6522795.

[34] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, CoRR abs/1705.09811 (2017).

[35] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012.

Constraints Propagation on GPU: A Case Study for AllDifferent[*]

Fabio Tardivo[1,*], Agostino Dovier[2,4], Andrea Formisano[2,4], Laurent Michel[3] and Enrico Pontelli[1]

[1] *New Mexico State University, Las Cruces, NM, USA*
[2] *CLPLab - DMIF - Università di Udine, Udine, Italy*
[3] *Department of Computer Science and Engineering, University of Connecticut, Storrs, CT, USA*
[4] *GNCS-INdAM, Gruppo Nazionale per il Calcolo Scientifico, Roma, Italy*

Abstract
The *AllDifferent* constraint is a fundamental tool in Constraint Programming. It naturally arises in many problems, from puzzles to scheduling and routing applications. Such popularity has prompted an extensive literature on filtering and propagation for this constraint. Motivated by the benefits that GPUs offer to other branches of AI, this paper investigates the use of GPUs to accelerate filtering and propagation. In particular, we present an efficient parallelization of the *AllDifferent* constraint on GPU; we analyze different design and implementation choices and evaluates the performance of the resulting system on medium to large instances of the Travelling Salesman Problem with encouraging results.

Keywords
Constraint Propagation, AllDifferent, Parallelism, GPU computing

1. Introduction

Constraint programming (CP) is a declarative paradigm to modeling and solving combinatorial problems. Users model a problem using a set of variables, each of them provided with a set of possible values (the domain of the variable), and a set of constraints that characterize the feasible solutions. Dedicated *constraint solvers* are used to process the problem models and identify solutions. Thanks to the MiniZinc Challenge [1], an annual competition among solvers, the constraint programming language MiniZinc [2] has emerged as a de-facto standard modeling language for the CP community.

Traditional constraint solvers work by alternating two stages: non-deterministic variables assignment and constraint propagation. Once a value has been assigned to a variable, constraint

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
[*] Research partially supported by Fondazione Friuli/Università di Udine project on Artificial Intelligence for Human Robot Collaboration and by projects INDAM GNCS-2020 *NoRMA* and INDAM GNCS-2022 InSANE (CUP_E55F22000270001). Laurent Michel was partially supported by the Synchrony Chair in Cybersecurity.
[*] Corresponding author.
ftardivo@nmsu.edu (F. Tardivo); agostino.dovier@uniud.it (A. Dovier); andrea.formisano@uniud.it (A. Formisano); ldm@uconn.edu (L. Michel); epontell@nmsu.edu (E. Pontelli)
0000-0003-3328-2174 (F. Tardivo); 0000-0003-2052-8593 (A. Dovier); 0000-0002-6755-9314 (A. Formisano); 0000-0001-7230-7130 (L. Michel); 0000-0002-7753-1737 (E. Pontelli)

propagation eliminates all values from domains of other variables that are incompatible in any solution with the assignment that has just been made. Alternative assignments are typically explored through backtracking.

The effectiveness of constraint propagation is heavily dependent on how the problem is modeled. For example, it is frequently possible to model the same problem using either a collection of elementary (e.g., binary or ternary) constraints or a single constraint involving many variables (i.e., a *global* constraint). Global constraints have the advantage of capturing a complex relationship between many variables, typically allowing a more extensive level of propagation. The impact of propagation on the structure of the search tree explored by a constraint solver can be significant—indeed, the propagation of global constraints is the subject of many studies and optimizations [3].

The *AllDifferent* constraint, which requires all variables in the constraint to be assigned a distinct value, naturally arises in many problems, from puzzles to scheduling and routing applications. Such popularity has prompted extensive studies on the propagation of this global constraint. There are different algorithms to propagate the *AllDifferent* constraint, each with a different trade-off between propagation strength and computational cost [4]. The most popular approach is the one by Régin [5].

Recently, branches of AI like Machine Learning have obtained huge benefits from the use of GPUs to speed up their tasks. Relatively more limited work has been done in exploring the use of GPUs for logic-based AI, e.g., [6] for SAT, [7] for ASP, and [8] for CP. For additional references, please see the recent surveys on parallelism in constraint and logic programming [9, 10, 11].

In this paper, we present a GPU-accelerated propagator for the *AllDifferent* constraint and its implementation within a simple constraint solver compatible with the MiniZinc language. Our contributions are: an analysis of Regin's algorithm to identify which parts are amenable of parallelization using a GPU; the comparison of alternative parallelization approaches; the integration of both the MiniZinc support and our GPU-accelerator propagator in a lightweight constraint solver [12]; and a comparison between the standard propagator and our GPU-accelerated version. Results on medium to large instances of the Travelling Salesman Problem demonstrate encouraging speedup.

The rest of the paper is organized as follows: Section 2 gives an introduction to CP, the Regin's algorithm for *AllDifferent* and the use of GPU for general computation. Section 3 describes the parallelization process, the implementation details of the final algorithm, and the integration in a constraint solver. In Section 4, we describe the benchmarks used to test the GPU-accelerated propagators and their results. Finally, Section 5 summarizes the paper and gives some directions for future works.

2. Background

2.1. Constraint Satisfaction Problem

A *Constraint Satisfaction Problem (CSP)* can be described by a triple $P = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{V} = \{V_1, \ldots, V_n\}$ is a finite set of variables, $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ is a finite set of sets, called *domains,* and $\mathcal{C}$ is a set of *constraints* on the variables $\mathcal{V}$. The domain $\mathcal{D}_i$ captures the allowable values for the variable V_i. Every *constraint* $c \in \mathcal{C}$ is defined over a subset $var(c) \subseteq \mathcal{V}$. Assume $var(c) = \{V_{i_1}, \ldots, V_{i_m}\}$, then c is a *relation* on $\mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_m}$, namely $c \subseteq \mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_m}$. A *solution* is an assignment $\sigma : \mathcal{V} \longrightarrow \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_n$ such that:

- for $i = 1, \ldots, n : \sigma(V_i) \in \mathcal{D}_i$ and
- for all c in $\mathcal{C}$, if $var(c) = \{V_{i_1}, \ldots, V_{i_m}\}$, then $\langle \sigma(V_{i_1}), \ldots, \sigma(V_{i_m}) \rangle \in c$.

In this paper, we focus on CSPs on finite domains, i.e., each $\mathcal{D}_i$ is a finite set. Whenever clear from the context, we will use syntactic sugars for commonly understood constraints (e.g., $V_3 < 2V_5$). We will use the term *global constraint* to refer to constraints that define relationships between a non-fixed number of variables.

Given a CSP P, a *constraint solver* looks for one or more solutions of P. A typical solver alternates two types of processes in the search for solutions: **(1)** constraint propagation and **(2)** non-deterministic choices. The latter step is used to select the next variable to be assigned and to select non-deterministically a value to be given to the variable (drawn from its current domain). Constraint propagation makes use of the constraints to remove from the domains of the variables values that can be proved not to belong to a solution. The choice of the variable is typically fast compared to the cost of constraint propagation.

During constraint propagation, constraints are placed in a queue for processing — i.e., to filter the domains of the variables involved in the constraints. A general property one could consider during propagation is *hyper-arc consistency* [3]. An m-ary constraint c on the variables $var(c) = \{V_{i_1}, \ldots, V_{i_m}\}$ is *hyper arc consistent (HAC)* if for all $j = 1, \ldots, m$ it holds that:

$$\begin{array}{l}(\forall a_j \in \mathcal{D}_{i_j})(\exists a_1 \in \mathcal{D}_{i_1}) \cdots (\exists a_{i-1} \in \mathcal{D}_{i_{j-1}}) \\ \qquad\qquad (\exists a_{i+1} \in \mathcal{D}_{i_{j+1}}) \cdots (\exists a_m \in \mathcal{D}_{i_m})(\langle a_1, \ldots, a_m \rangle \in c)\end{array}$$

A CSP is *hyper-arc consistent* if all constraints in $\mathcal{C}$ are HAC. In case of binary constraints (i.e., $m = 2$) the HAC property reduces to *arc consistency.*

The complexity of naive algorithms for obtaining HAC is exponential in m. It is also common practice to simplify constraints involving many variables into collections of constraints involving a smaller number of variables (e.g., 2 or 3). For example, a constraint like $X + 2Y + 3U < 4V + Z$ can be translated to $A < B, A = X + C, C = 2Y + 3U, B = 4V + Z$. However, this type of translations may lead to a reduced filtering capability during constraint propagation, since the HAC property is guaranteed only for the simple constraints. In the example above, a built-in constraint capable of handling sums of linear terms can be more efficient. We can first rewrite the constraint as $X + 2Y + 3U - 4V - Z < 0$ and then use the scalar product to write it equivalently as: $[X, Y, U, V, Z] \cdot [1, 2, 3, -4, -1] < 0$.

The constraint programming literature has explored a number of dedicated algorithms to handle propagation for specific types of constraints. In this work, we focus on the global constraint *AllDifferent*.

2.2. AllDifferent

The *AllDifferent* global constraint deals with a list of variables (of any length) and aims at ensuring that all of them are assigned pairwise different values in the solution. Even though *AllDifferent*$(x_1, \ldots, x_n)$ admits exactly the same set of solutions as the set of binary constraints

$\{x_i \neq x_j \,:\, 1 \leq i < j \leq n\}$, arc consistency applied to the individual binary constraints delivers a weaker filtering of the domains than considering the original global constraint (see, e.g., [4]).

Régin's well-known algorithm [5] for *AllDifferent* is based on a bipartite graph representation of the constraint that matches variables with values. In general, a bipartite graph $G(N_1 \cup N_2, E)$, is defined over two disjoint sets of vertices N_1 and N_2 and $E \subseteq \{\{u, v\} \,:\, u \in N_1, v \in N_2\}$ are undirected edges. A *matching* of a bipartite graph is a set of edges $M \subseteq E$ such that no two distinct edges share a vertex. A maximum matching is a maximum cardinality matching. The Hopcroft–Karp algorithm [13] for computing a maximum matching in a bipartite graph has $O(\sqrt{n} \cdot |E|)$ running time, while the Ford–Fulkerson algorithm, which reduces the problem to a maximum flow, has time complexity $O(n \cdot |E|)$ [14], where $n = |N_1| + |N_2|$.

A *directed graph (digraph)* $G(N, A)$ pairs a set of vertices N with a set of arcs $A \subseteq N \times N$, i.e., a set of directed edges. A *path* $x_0, x_1, \ldots, x_m$ is a sequence of vertices such that $(x_i, x_i + 1) \in A$ for $i = 0, \ldots, m - 1$. If $x_m = x_0$ the path is called a cycle. A *Strongly Connected Component (SCC)* M of G is a maximal subset of N such that, for all pairs $u, v \in M$, there is a path $u = x_0, x_1, \ldots, x_m = v$. It follows that there are no cycles with edges between different SCCs. The set of SCCs forms a partition of the vertices of the digraph. Tarjan's algorithm can be used to efficiently compute the SCCs of any digraph in $O(|N| + |A|)$ time [15].

Before discussing the GPU-based implementation, it is perhaps useful to briefly review the steps adopted in the propagation for the *AllDifferent* constraint. In particular, consider the constraint applied to n variables, i.e.

$$\mathit{AllDifferent}(x_1, \ldots, x_n)$$

Consider the following preliminary definitions. Given a bipartite graph $G(N_1 \cup N_2, E)$ and a matching M of G, the *residual graph* from G and M is a directed graph $R(N_R, A_R)$ built as follows (see Figure 1):

1. The matching M is used to define the set of arcs A_1 that directs the edges of E

$$\begin{array}{rcl} A_1 & = & \{(x, d) \,:\, x \in N_1, d \in N_2, \{x, d\} \in E \setminus M\} \cup \\ & & \{(d, x) \,:\, x \in N_1, d \in N_2, \{x, d\} \in M\} \end{array}$$

   Namely, for each matching edge, there is an arc from value to variable and for each non-matching edge, the arc is directed from variable to value.
2. A new sink node $t \notin N_1 \cup N_2$ is added and $N_R = N_1 \cup N_2 \cup \{t\}$.
3. The matching M is used to define the set of arcs between t and the nodes in N_2

$$\begin{array}{rcl} A_2 & = & \{(d, t) \,:\, d \in N_2, (\nexists x \in N_1)(\{d, x\} \in M)\} \cup \\ & & \{(t, d) \,:\, d \in N_2, (\exists x \in N_1)(\{d, x\} \in M\} \end{array}$$

4. Finally, the set of arcs A_R is defined as $A_R = A_1 \cup A_2$

Let us now review the algorithm to propagate *AllDifferent*$(x_1, \ldots, x_n)$. The algorithm constructs a bipartite graph $G = (N_1 \cup N_2, E)$ where:

- $N_1 = \{x_1, \ldots, x_n\}$,

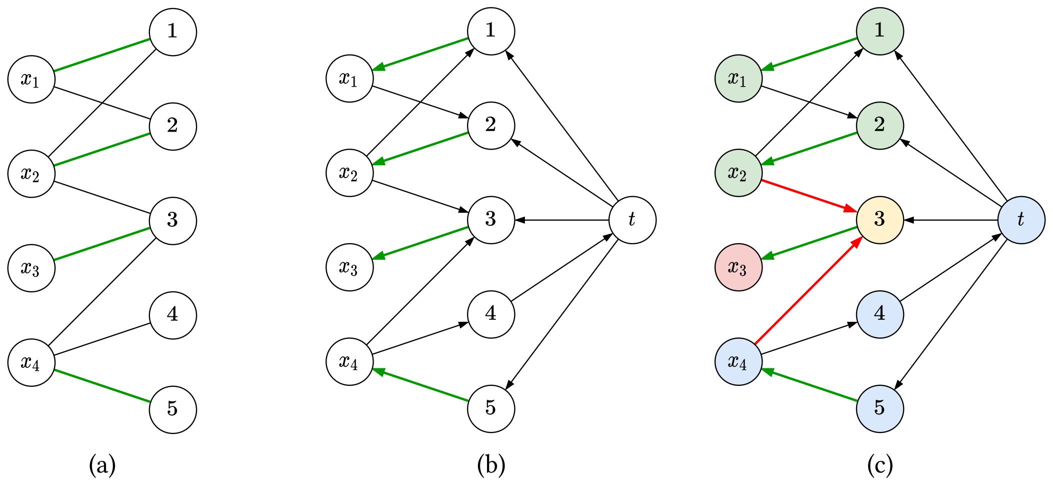


Figure 1: Quick overview of Regin's algorithm on x_1, x_2, x_3, x_4 where $D_1 = \{1, 2\}$, $D_2 = \{1, 2, 3\}$, $D_3 = \{3\}$, $D_4 = \{3, 4, 5\}$. In (a) is highlighted the maximum match of step 1. In (b) is pictured the residual graph of step 3. In (c) are highlighted the SCCs of step 4, each with a different color, and in red the arcs considered in step 5.

- $N_2 = \bigcup_{i \in 1..n} \mathcal{D}_i$, where $\mathcal{D}_i$ is the domain of the variable x_i, and
- $E = \{\{x_i, d\} \mid i \in 1..n \wedge d \in \mathcal{D}_i\}$

The algorithms proceeds as follows (see also Figure 1)

1. Find a maximum matching M for $G(N_1 \cup N_2, E)$.
2. If $|M| < n$, then the constraint is unsatisfiable
3. Otherwise, construct the *residual digraph* $R(N_R, A_R)$ from G and M.
4. Compute the strongly connected components of R.
5. For every variable x_i, remove from its domains all the values d such that there exists an arc $(x_i, d) \in A_R$ or $(d, x_i) \in A_R$ that is not in M and connects two distinct SCCs.

In our implementation we use the Hopcroft-Karp's algorithm for step 1, with a time complexity $O(\sqrt{|N_1| + |N_2|} \cdot |E|)$. Step 2 has complexity $O(1)$ since is just a check. Step 3 has complexity $O(|N_1| + |N_2| + |E|)$ as described below. In step 4 we use the Tarjan's algorithm with complexity $O(|N_1| + |N_2| + |A|)$. Finally, step 5 has time complexity $O(|A|)$ since it scans all the arcs. In practice, the computational time can be reduced using several optimizations [16]. Our implementation mitigates the cost of step 1 using an incremental approach as is traditionally done.

Correctness of the procedure follows from a theorem by Berge that characterize the edges that belongs to some but not to all maximum matchings by just analyzing a single maximum matching [17].

2.3. GPGPU with CUDA

General-Purpose computing on Graphics Processing Units (GPGPU) is the use of a Graphics Processing Unit (GPU) to speed up computations traditionally handled by the Central Processing

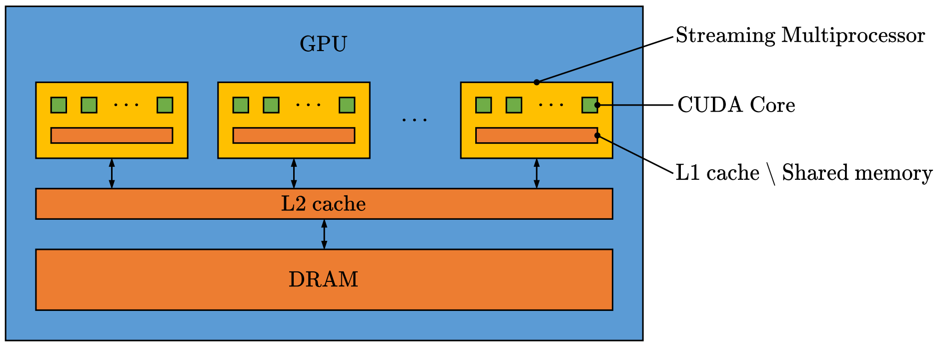


Figure 2: Simplified GPU architecture

Unit (CPU). NVIDIA introduced the *Compute Unified Device Architecture (CUDA),* a general-purpose programming library that allows programmers to ignore the underlying graphical concepts in favor of high-performance computing concepts [18]. CUDA has been successfully used to accelerate computations in a variety of domains, such as physics, bioinformatics, and machine learning [19].

The architecture of a GPU (Figure 2) includes a main memory (DRAM), typically off-chip and used as global memory, an L2 cache, and an array of *Streaming Multiprocessors (SM).* Each SM contains a small amount of on-chip fast memory, used as L1 cache or scratchpad memory (the *Shared memory*), and several *CUDA cores,* responsible for the actual execution of instructions. The architecture is designed to enable rapid context switching of lightweight threads.

The underlying conceptual model for parallelism supported by CUDA is *Single-Instruction Multiple-Thread (SIMT),* (variant of SIMD) where the same instruction is executed by different threads, while data and operands may differ from thread to thread. A CUDA program includes parts meant for execution on the CPU (the *host*) and parts meant for parallel execution on the GPU (the *device*). Typically, the host code transfers data to the device memory (DRAM in Fig. 2), starts parallel computations on the device, and retrieves the results from device memory. The CUDA library supports interaction, synchronization, and communication between host and device. Each device computation is described as a collection of concurrent threads, each executing the same function (called a *kernel,* in CUDA terminology). Each thread is executed by a CUDA core; these threads are hierarchically organized in *blocks* of threads, assigned to SMs. The threads in a block assigned to an SM execute the same instruction on different data. In case of control flow divergence among the threads within a block, their execution is serialized. Device global memory is accessible by all threads, whereas threads of the same block may access the high-throughput shared memory.

3. Design and Implementation

In this section we explore the development of a constraint solver which supports parallel propagation of *AllDifferent* on GPUs.

The first step in this process consists of selecting an existing constraint solver suitable to host a GPU-enabled *AllDifferent*. Initially, we had a look at the fastest solvers compatible with the MiniZinc language and, thus, we selected OR-Tools [20], JaCoP [21], and Gecode [22] respectively Gold and Silver medal of the last MiniZinc challenge [23]. We realized soon that their efficiency is also due to several optimizations that makes the code difficult to modify.

Our choice converged on *MiniCP* [12], a lightweight solver specifically designed to enable research and exploration of diverse search and propagation methodologies. MiniCP provides a comprehensive documentation and a clean design. In particular, our research builds on *MiniCPP* [24], a C++ implementation of MiniCP, suitable to host C++ CUDA kernels. With minor optimizations, MiniCPP provides a performance that is comparable to that of other solvers (e.g., JaCoP) for several classes of problems.

To use MiniCPP as a base solver, it was necessary to implement the support for the MiniZinc language, using the FlatZinc skeleton parser provided by Gecode, and implement all the standard integer and Boolean constraints. Moreover, we created the necessary definitions to allow the MiniZinc compiler to recognize the MiniCPP's native *AllDifferent* as a global constraint, thus avoiding its decomposition in a collection of binary constraints.

The following subsection describes the implementation of the parallel version of the *AllDifferent* filtering algorithm on GPU and how it has been integrated in the solver. We enabled the FlatZinc parser to recognize the annotation ::gpu to instruct the solver to propagate the annotated constraint using the GPU algorithm in place of the standard CPU algorithm.

3.1. Parallelization

The key components of the filtering algorithm for *AllDifferent* propagation are *(1)* the computation of a Maximum Matching in a bipartite graph (MM) and *(2)* the computation of the Strongly Connected Components (SCC) of a directed graph (see Section 2.2).

Before discussing the parallelization process we introduce some preliminary definitions. *Breadth-First Search* (BFS) is a graph traversal algorithm that explores the graph's vertices in the order of their distance from a source vertex s. Given a graph $G = (V, E)$ and a source vertex $s \in V$, BFS systematically traverses the edges in such a way that all vertices at distance k from s are discovered before any vertices at distance $k + 1$. By BFS is possible to find all the $v \in V$ reachable from s. In case of digraph, the *forward reachability* of a vertex s is defined as the set of nodes F such that exists a path from s to any $v \in F$. Similarly the *backward reachability* of a vertex s is the set of nodes B such that for any $v \in B$ exists a path from v to s. Forward / backward reachability can be expressed as a binary matrix where the element at coordinates (i, j) is 1 if and only if vertex i reaches / is reachable from vertex j.

Computing a Maximum Matching on an GPU. There are several approaches to solving such problems on GPU. For maximum matching there are implementations based on the auction [25], push-relabel [26], and the BFS [27] algorithms.
The auction algorithm works as an auction where persons compete for objects by raising their prices. It alternates bidding and assignment phases until all the person have been assigned to an object. The bidding and assignment phases are offloaded on the GPU, where bids and assignments are computed in parallel.
The push-relabel approach solves the maximum matching reducing it to a flow problem. The

initial bipartite graph $G = (N_1 \cup N_2, E)$ is modified by adding two nodes s and t, the first connected to all the $n \in N_1$ and the second reached by all the $n \in N_2$. The resulting graph is seen as a flow network such that:

- through an edge can pass 0 or 1 unit of flow
- the node s produces N_1 units of flow and the node t can receive N_1 units of flow
- the sum of the ingoing flow in a node must be equal to the sum of the outgoing flows

The problem is to find which edges to use to move the maximum amount of flow from s to t. The push-relabel algorithm alternates push operations where flow is pushed through an edge, and relabel operation to mark the nodes with an excess of ingoing flow. Such alternation is repeated until no nodes, except t, have an excess of ingoing flow. The push and relabel phases are offloaded to the GPU where each node is processed in parallel.
The BFS algorithms are based on the Hopcroft-Karp algorithm and make use of a GPU-accelerated parallel BFS to find the augmenting paths in place of the standard Depth First Search.

We started our study with a push-relabel algorithm based on [26]. We choose it because its more studied than the other approaches and it does not assume the existence of a matching. Despite our optimization efforts, offloading the calculation of MM on the GPU does not pay off: the algorithm does not scale [28] and is slower than on the CPU since the solvers can quickly calculate MM incrementally [16]. Moreover, for large instances, the cost of MM is negligible compared with the cost of SCCs. In the end, the computation of the maximum matching was kept on the CPU.

Strongly Connected Components on a GPU. For SCCs, the majority of the GPU implementations [29, 30] ultimately make use of forward/backward reachability [31]. The literature about SCCs on GPU considers enormous sparse graphs with millions of nodes. The trend is to use, as a fundamental step, a parallel BFS to calculate forward/backward reachability. This scenario does not fit our context where (1) a major constraint leads to a dense graph of hundreds/thousands of nodes and (2) we aim for low latency and BFS notoriously suffers from load imbalance [32]. The first observation direct us to calculate SCCs using forward reachability as follows.

Let A be the adjacency (binary) matrix of the graph, namely $A(i, j) = 1$ iff there is an edge between node i and node j. Then:

1. Compute the forward reachability matrix F from A.
2. Transpose F to obtain the backward reachability matrix B.
3. Create a matrix C such that $C(i, j) = F(i, j) \cdot B(i, j)$. That is, $C(i, j) = 1$ if and only if there is a cycle containing node i and node j.
4. The identifier of the SCC of the node i is the minimum j such that $C(i, j) = 1$.

The second observation made us look for alternatives to BFS. Let $G(V, E)$ be a graph, the standard algorithms to calculate the reachability matrix are:

- Matrix multiplication, with complexity $O(|V|^{2.8} log_2(|V|))$ [33].
- Warshall algorithm, with complexity $O(|V|^3)$ [34].

We explored the parallelization of both of these approaches (see Section 3.2) and chose Warshall algorithm.

3.2. Implementation Details

Data transfer. To begin with, we had to decide which data to transfer to the GPU. We opted to transfer the residual graph as a binary adjacency matrix. Preliminary tests showed that it is better than transferring only the domains and the match necessary to generate the graph's adjacency matrix. This is because, in our case, most of the transmission's cost is in the initialization phase than in the actual data transfer. Moreover, the build of the adjacency matrix requires many sparse memory accesses, and the CPU is sensibly faster than the GPU for such tasks.

Matrix representation. We choose to represent the adjacency matrix as a bit matrix for its several benefits. It enables the use of bitwise operations, it can be initialized by dumping the domains' internal representation, and it minimizes the amount of data to transfer. In preliminary tests, we also explored the use of other representations suited for hardware accelerated matrix multiplication (i.e., Tensor Core), and for sparse matrices. In both cases, the matrix multiplication performs worst than an ad-hoc matrix multiplication encoded by us using bitwise operations. In the first case, the penalty come from the more general algorithm and dealing with floating-point numbers, while in the second case it comes from the fast increasing density of the matrices and dealing with integer numbers.

Matrix multiplication. After the choice of the representation, we focused on matrix multiplication. We tested two of the state-of-the-art GPU linear algebra libraries, namely cuBLAS [35] and cuBool [36]. The performances were, at best, matching the classic algorithm on CPU.

Due to these poor results, we focused on binary matrix multiplication on GPU. There are efficient implementations for multi-GPUs matrix multiplication [37] and to speedup Binarized Neural Network [38], but their code is tailored to their use case. Thus, we decided to implement binary matrix multiplication from scratch. We used a few known techniques in the design:

- tiled matrix multiplication to optimize cache usage
- data arrangement to optimize memory access, and
- bitwise operation to speed up the computation

The result was a simple but efficient binary matrix multiplication, with performances slightly worst than [37] for squared matrices of a few thousand rows.

Warshall. Then, we focused on the Warshall algorithm. Despite sharing a similar nested loops structure with the matrix multiplication, its nesting order is stricter and does not allows the same optimizations. The majority of the GPU implementations that use an adjacency matrix representation [39, 40] are based on a blocked version of the Warshall algorithm from [41]. Such an algorithm was developed to maximize the CPU's cache utilization, and it is particularly efficient to exploit the GPU's shared memory. As the Warshall algorithm, this blocked version starts from an adjacency matrix of size $n \times n$ and iteratively updates it to obtain the reachability matrix. It begins by dividing the matrix in tiles of size $t \times t$ and then performing n/t steps. Each step s is made of three phases (see Figure 3), each one updating a specific set of tiles according to their dependency:

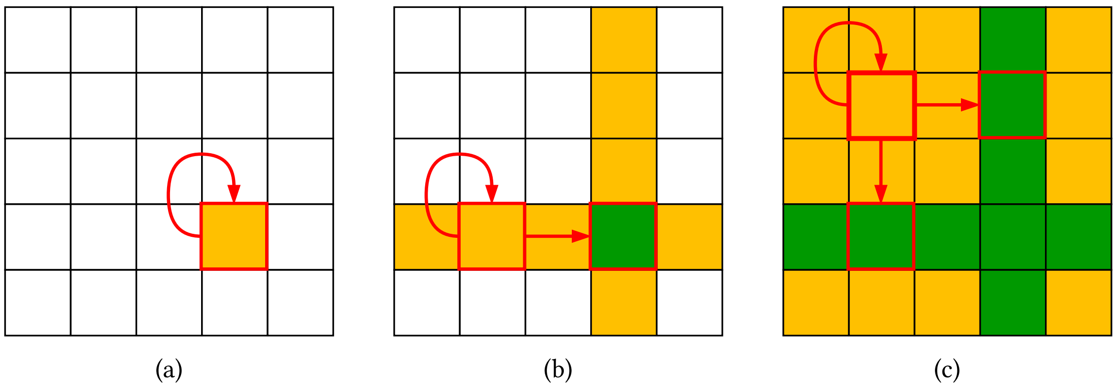


Figure 3: Illustration of the 4-th step of the blocked Warshall algorithm on a matrix divided in 25 tiles. The tiles updated in each phase are highlighted in yellow, while the tiles already processed are colored in green. Tiles dependencies are illustrated in with red arrows.

Phase 1 : This phase consider the s-th tile of the main diagonal, named D, and update it using the equation $D(i,j) = D(i,j) \vee (D(i,k)) \wedge D(k,j))$ for $0 \leq k < t$.

Phase 2 : This phase consider the tiles of the s-th row and s-th column excluding the D tile. A generic tile of the s-th row, named R, is updated using the equation $R(i,j) = R(i,j) \vee (D(i,k)) \wedge R(k,j))$. A tile of the s-th column, named C, is updated using the equation $C(i,j) = C(i,j) \vee (C(i,k)) \wedge D(k,j))$.

Phase 3 : This phase consider all the remaining tiles. The equation to update a generic tile T in position (r,c) is $T(i,j) = T(i,k) \vee (R(i,k)) \wedge C(k,j))$ where R is the tile in position (r,s) an C is the tile in position (s,c).

Each tile within a phase can be updated independently, allowing parallelization of phases 2 and 3. These independent updates map well into the GPU computational model, where each tile of size $t \times t$ is managed by a block of t threads. In this way, the i-th thread updates the i-th row of the tile considering every $0 \leq k < t$. We paid particular attention to optimizing phase 3 since it involves most of the tiles. Unlike phase 2, where the update of the i-th row $R(i,*)$ depends on the k-th row $R(k,*)$, in phase 3, the update of the i-th row $T(i,*)$ does not depends on other lines of the tile. This fact makes possible to avoid threads synchronization, sensibly reducing the computational time. Preliminary benchmarks show that the most convenient tile size is $t = 128$. Such dimension allows reading each row in one memory access as one `uint4` (32*4 bit) and manipulating it by two 64-bit operations. Preliminary comparisons with the matrix multiplication approach reported similar performance. However, in such tests, the matrix multiplication approach performed only a few of the $log_2(|V|)$ iterations. In the end, we chose as our approach the blocked Warshall algorithm because its runs in a slightly less amount of time as a good case of the matrix multiplication approach.

Initial tests highlight that offloading the computation on GPU is not convenient when the bipartite graphs are small (i.e. $|V| \leq 200$). For these cases we created a single procedure that performs steps 1, 4 without performing steps 2, 3.

4. Results and Analysis

It is expected that a GPU implementation would provide benefits on instances that are sufficiently large, as the setup overhead would otherwise overshadow the benefits of the parallel execution. We chose as benchmark the Travelling Salesman Problem (TSP) because it can be simply modeled using a Circuit constraint, that internally makes use of the *AllDifferent* constraint, there is an established set of large benchmarks [42], and it is a fundamental problem for routing applications.

We select about 80 instances from the TSPLib with 100 to 10,000 cities and convert them into the MiniZinc format. We solve them using the MiniCP's native *AllDifferent* propagator as well as our GPU-accelerated propagator. All benchmarks have a timeout of 10 minutes and are executed on a system equipped with an Intel Core i7-10700K, 32GB of DDR4 RAM, and GeForce RTX 3080 running Ubuntu 21.04 and CUDA 11.4.

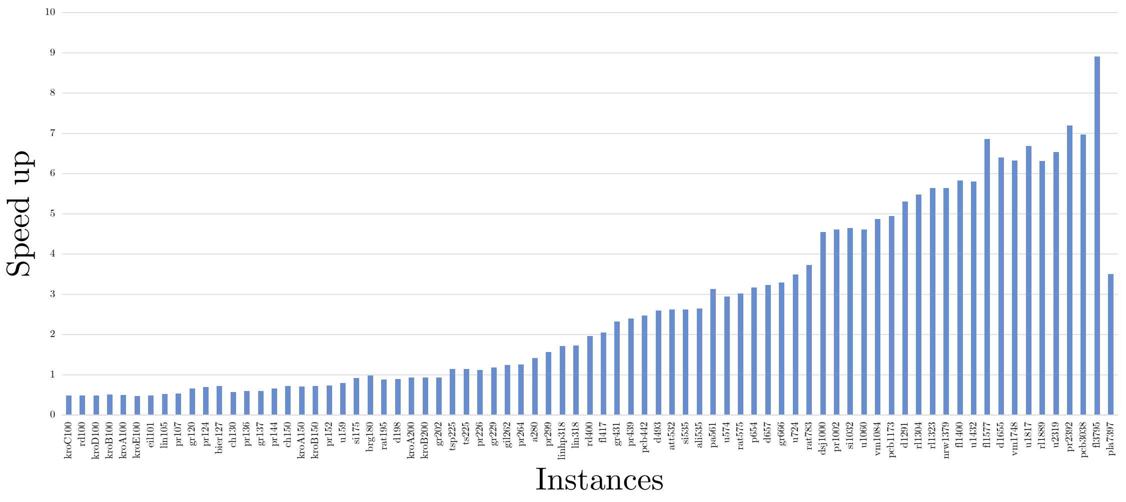


Figure 4: Results of the TSPLib benchmarks. On the horizontal axis there are the instances sorted by increasing size. The vertical values indicate the speedup of the GPU in terms of explored nodes.

Figure 4 illustrates the results of the benchmarks. The speedup is calculated as ratio between the GPU search speed and the CPU search speed. The search speed is the number of visited nodes over the search time. As expected from preliminary tests, the benefits of our approach start to be visible when a constraint involves several hundreds of variables. The plot shows an increasing speedup as the size of the instance increases, except for the largest instance. Such behavior is due to the large time between two GPU-accelerated propagations, reducing the opportunities to speed up the computation. To verify such explanation we increased the timeout to 30 minutes and obtained a speedup of approximately 7 times.

5. Conclusion and Future Works

Motivated by the benefits that GPUs offer in terms of computational power, we designed and implemented a GPU-accelerated propagator for the *AllDifferent* constraint. We described the process of developing such a propagator, which challenges we encountered, and the motivations behind the main implementation choices. The propagator has been integrated into an existing solver. We tested our implementation on medium to large instances of the Travelling Salesman Problem and obtained speedups up to 7 times in terms of explored nodes. Unlike other parallel approaches, our method is immediately usable since modern PCs often have a GPU.

There are many ways to extend and improve this work: implementing on GPU propagators for other global constraints, exploring their usage in Constraint-Based Local Search, etc. Our next step will be focusing on problems containing multiple *AllDifferent* constraints. In such cases it is possible to process the *AllDifferent* constraints in parallel. This promises significant speedups even for small to medium problems, where the GPU propagation is still less efficient than the CPU version.

References

[1] P. J. Stuckey, R. Becket, J. Fischer, Philosophy of the MiniZinc challenge, Constraints 15 (2010) 307–316. doi:10.1007/s10601-010-9093-0.

[2] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, G. Tack, MiniZinc: Towards a standard CP modelling language, in: Principles and Practice of Constraint Programming – CP 2007, volume 4741 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 529–543. doi:10.1007/978-3-540-74970-7_38.

[3] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006. doi:10.1016/s1574-6526(06)x8001-x.

[4] W. J. van Hoeve, The alldifferent constraint: A survey, 2001. URL: https://arxiv.org/abs/cs/0105015.

[5] J.-C. Régin, A filtering algorithm for constraints of difference in CSPs, in: Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1), AAAI '94, American Association for Artificial Intelligence, USA, 1994, p. 362–367.

[6] A. Dal Palù, A. Dovier, A. Formisano, E. Pontelli, CUD@SAT: SAT solving on GPUs, Journal of Experimental & Theoretical Artificial Intelligence 27 (2014) 293–316. doi:10.1080/0952813x.2014.954274.

[7] A. Dovier, A. Formisano, E. Pontelli, F. Vella, A GPU implementation of the ASP computation, in: Practical Aspects of Declarative Languages, Springer International Publishing, 2016, pp. 30–47. doi:10.1007/978-3-319-28228-2_3.

[8] F. Campeotto, A. D. Palù, A. Dovier, F. Fioretto, E. Pontelli, Exploring the use of GPUs in constraint solving, in: Practical Aspects of Declarative Languages, Springer International Publishing, 2014, pp. 152–167. doi:10.1007/978-3-319-04132-2_11.

[9] I. P. Gent, I. Miguel, P. Nightingale, C. Mccreeh, P. Prosser, N. C. A. Moore, C. Unsworth, A review of literature on parallel constraint solving, Theory and Practice of Logic Programming 18 (2018) 725–758. doi:10.1017/s1471068418000340.

[10] A. Dovier, A. Formisano, E. Pontelli, Parallel answer set programming, in: Handbook of Parallel Constraint Reasoning, Springer International Publishing, 2018, pp. 237–282. doi:10.1007/978-3-319-63516-3_7.

[11] A. Dovier, A. Formisano, G. Gupta, M. V. Hermenegildo, E. Pontelli, R. Rocha, Parallel

logic programming: A sequel, Theory and Practice of Logic Programming (2022) 1–69. doi:10.1017/s1471068422000059.

[12] L. Michel, P. Schaus, P. V. Hentenryck, MiniCP: a lightweight solver for constraint programming, Mathematical Programming Computation 13 (2021) 133–184. doi:10.1007/s12532-020-00190-7.

[13] J. E. Hopcroft, R. M. Karp, A n^5/2 algorithm for maximum matchings in bipartite graphs, in: 12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13-15, 1971, IEEE Computer Society, 1971, pp. 122–125. doi:10.1109/SWAT.1971.1.

[14] L. R. Ford, D. R. Fulkerson, Maximal flow through a network, Canadian Journal of Mathematics 8 (1956) 399–404. doi:10.4153/cjm-1956-045-5.

[15] R. Tarjan, Depth-first search and linear graph algorithms, SIAM Journal on Computing 1 (1972) 146–160. doi:10.1137/0201010.

[16] I. P. Gent, I. Miguel, P. Nightingale, Generalised arc consistency for the AllDifferent constraint: An empirical survey, Artificial Intelligence 172 (2008) 1973–2000. doi:10.1016/j.artint.2008.10.006.

[17] C. Berge, Graphs and Hypergraphs, Elsevier Science Ltd., GBR, 1985.

[18] NVIDIA Team, CUDA toolkit documentation, (n.d.). URL: https://developer.nvidia.com/cuda-zone.

[19] NVIDIA Team, GPU accelerated applications, (n.d.). URL: https://www.nvidia.com/en-us/gpu-accelerated-applications.

[20] L. Perron, V. Furnon, OR-Tools, (n.d.). URL: https://developers.google.com/optimization/.

[21] K. Kuchcinski, R. Szymanek, JaCoP, (n.d.). URL: https://github.com/radsz/jacop.

[22] Gecode Team, GECODE, (n.d.). URL: https://www.gecode.org.

[23] MiniZinc Team, The MiniZinc challenge, (n.d.). URL: https://www.minizinc.org/challenge.html.

[24] R. Gentzel, L. Michel, W.-J. van Hoeve, HADDOCK: A language and architecture for decision diagram compilation, in: Lecture Notes in Computer Science, Springer International Publishing, 2020, pp. 531–547. doi:10.1007/978-3-030-58475-7_31.

[25] C. N. Vasconcelos, B. Rosenhahn, Bipartite graph matching computation on GPU, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 42–55. doi:10.1007/978-3-642-03641-5_4.

[26] J. Wu, Z. He, B. Hong, Efficient CUDA algorithms for the maximum network flow problem, in: GPU Computing Gems Jade Edition, Elsevier, 2012, pp. 55–66. doi:10.1016/b978-0-12-385963-1.00005-8.

[27] M. Deveci, K. Kaya, B. Uçar, Ü. V. Çatalyürek, GPU accelerated maximum cardinality matching algorithms for bipartite graphs, in: F. Wolf, B. Mohr, D. an Mey (Eds.), Euro-Par 2013 Parallel Processing, volume 8097 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 850–861. doi:10.1007/978-3-642-40047-6_84.

[28] P. M. Jensen, N. Jeppesen, A. B. Dahl, V. A. Dahl, Review of serial and parallel min-cut/max-flow algorithms for computer vision, 2022. URL: https://arxiv.org/abs/2202.00418.

[29] J. Barnat, P. Bauch, L. Brim, M. Ceška, Computing strongly connected components in parallel on CUDA, in: 2011 IEEE International Parallel & Distributed Processing Symposium, IEEE, 2011, pp. 544–555. doi:10.1109/ipdps.2011.59.

[30] G. Li, Z. Zhu, Z. Cong, F. Yang, Efficient decomposition of strongly connected components on GPUs, Journal of Systems Architecture 60 (2014) 1–10. doi:10.1016/j.sysarc.2013.10.014.

[31] L. K. Fleischer, B. Hendrickson, A. Pınar, On identifying strongly connected components in parallel, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, pp. 505–511. doi:10.1007/3-540-45591-4_68.

[32] H.-N. Tran, E. Cambria, A survey of graph processing on graphics processing units, The Journal of Supercomputing 74 (2018) 2086–2115. doi:10.1007/s11227-017-2225-1.

[33] M. J. Fischer, A. R. Meyer, Boolean matrix multiplication and transitive closure, in: 12th Annual Symposium on Switching and Automata Theory (swat 1971), IEEE, 1971, pp. 129–131. doi:10.1109/swat.1971.4.

[34] S. Warshall, A theorem on Boolean matrices, Journal of the ACM 9 (1962) 11–12. doi:10.1145/321105.321107.

[35] NVIDIA Team, cuBLAS, (n.d.). URL: https://developer.nvidia.com/cublas.

[36] E. Orachyov, P. Alimov, S. Grigorev, cuBool: sparse Boolean linear algebra for Nvidia Cuda, (n.d.). URL: https://github.com/JetBrains-Research/cuBool.

[37] M. Karppa, P. Kaski, Engineering boolean matrix multiplication for multiple-accelerator shared-memory architectures, CoRR abs/1909.01554 (2019). URL: http://arxiv.org/abs/1909.01554.

[38] A. Li, S. Su, Accelerating binarized neural networks via bit-tensor-cores in Turing GPUs, CoRR abs/2006.16578 (2020). URL: https://arxiv.org/abs/2006.16578.

[39] G. J. Katz, J. T. K. Jr., All-pairs shortest-paths for large graphs on the GPU, in: D. Luebke, J. Owens (Eds.), Proceedings of the EUROGRAPHICS/ACM SIGGRAPH Conference on Graphics Hardware 2008, Sarajevo, Bosnia and Herzegovina, 2008, Eurographics Association, 2008, pp. 47–55. doi:10.2312/EGGH/EGGH08/047-055.

[40] K. Matsumoto, N. Nakasato, S. G. Sedukhin, Blocked all-pairs shortest paths algorithm for hybrid CPU-GPU system, in: 2011 IEEE International Conference on High Performance Computing and Communications, IEEE, 2011, pp. 145–152. doi:10.1109/hpcc.2011.28.

[41] G. Venkataraman, S. Sahni, S. Mukhopadhyaya, A blocked all-pairs shortest-paths algorithm, ACM Journal of Experimental Algorithmics 8 (2003). doi:10.1145/996546.996553.

[42] G. Reinelt, TSPLIB—a traveling salesman problem library, ORSA Journal on Computing 3 (1991) 376–384. doi:10.1287/ijoc.3.4.376.

A Framework to build Abductive-Deductive Chatbots, based on Natural Language Processing and First-Order Logic

Carmelo Fabio Longo[1], Corrado Santoro[1]

[1]Department of Mathematics and Computer Science, University of Catania, Viale Andrea Doria, 6 - 95125 - Catania, IT

Abstract
This paper presents a framework based on natural language processing and first-order logic, which implicitly simulate the human brain features of selecting properly information related to a query from a knowledge base (abductive pre-stage), before to infer new knowledge from such a selection acting as deductive database. Such features are used with the aim of instantiating *cognitive* chatbots, able of human-like fashioned reasoning, supported by a module which automatically transforms polar and wh-questions into one or more likely assertions, in order to infer Boolean values or snippets with variable length as factoid answer from a *conceptual* knowledge base. The latter is splitted into two layers, representing both long- and short-term memory, and the transition of information between the two layers is achieved leveraging both a greedy algorithm and the engine's features of a NoSQL database, with promising timing performance than respect using one layer. Furthermore, such chatbots don't need any scripts updates or code refactory when new knowledge has to income, but just the knowledge itself in natural language.

Keywords
Artificial Intelligence, Chatbot, First-Order Logic, Cognitive Architecture, Deductive Database

1. Introduction

Nowadays, the momentousness which tech giants like Google, Meta (former Facebook), Microsoft, IBM and Amazon are giving to chatbots, is a strong indicator that this technology will play a significant role in the future. Among applications leveraging Natural Language Processing (NLP), those related to chatbots systems are growing very fast and present a wide range of choices depending on the usage, each with different complexity levels, expressive powers and integration capabilities. At the present, if you want to know trending movies in your area, you could use the Fandango Bot[1]; or, if you want to get NBA highlights and updates, you could use NBA's bot[2], and so forth. On the other hand, the way towards a human-like fashioned reasoning chatbot is quite long and challenging. Such an ideal agent-chatbot, in addition to having deductive, abductive and inductive capabilities, should be able of commonsense categorization, even making usage of the so-called *Frames* [1] and *Scripts* [2], and so on. In

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
fabio.longo@unict.it (C. F. Longo); santoro@dmi.unict.it (C. Santoro)
0000-0002-0877-7063 (C. F. Longo); 0000-0001-7116-9338 (C. Santoro)

[1]https://www.facebook.com/Fandango/
[2]https://www.facebook.com/nba/

order to achieve that, the cognitive disciplines involving has become a mandatory step in the overall process of such agent's modelling, together with knowledge of the linguistic science.

In this work, which is somehow the evolution of [3] where we showed the effectiveness of a similar approach in the case of automation commands provided in natural language, we present a framework called AD-Caspar based on NLP and First-Order Logic (FOL), as baseline platform for implementing scalable and flexible *cognitive* chatbots with both goal-oriented and conversational features. A first overview of such an architecture has been presented here [4]. The first prototype of AD-Caspar is not yet provided with tools to build complex dialog systems, but differently from other platforms, in order to handle additional question-answer couples, the user has to provide just the related sentences in natural language, without the need of updating the chatbot code at design-time. After the agent has parsed every sentence, a FOL representation ot them is asserted in its conceptual KB, which will be able to act as a deductive database [5]. AD-Caspar inherits most of its features directly from its predecessor, namely Caspar [6], whose name stands for: *Cognitive Architecture System Planned and Reactive*. Caspar was designed to build goal-oriented agents (vocal assistants) with enhanced deductive capabilites, working on Internet of Things (IoT) scenarios. The additional features introduced in AD-Caspar, where AD- stands for: *Abductive Deductive*, are the usage of abduction as pre-stage of deduction, in order to make inferences only on a narrow set of query-related clauses, plus the application of question-answering techniques to deal with wh-questions and give back factoid answers (single nouns or snippets) in the best cases; otherwise, optionally, only a relevance-based output will be returned.

This paper is structured as follows: Section 2 is about the related literature of the typical approaches of chatbot applications; Section 3 shows in detail all the architecture's components and underlying modules, referring sometimes the reader to legacy-related literature for the sake of shortness; Section 4 shows how AD-Caspar deals with polar and wh-questions; Section 5 is about a case-study where it is shown a typical instance of a Telegram chatbot, working on small KBs; Section 6 summarizes the content of the paper and provides our conclusions, together with some future work perspective.

A Python prototype implementation of AD-Caspar is also provided for research purposes in a Github repository[3].

2. Related Work

In the plethora of known chatbot systems, first of all stand out the ones participating to the Loebner Prize Competition[4]. Typical sentences to prove Loebner price candidates effectiveness do not require particular logical inference, but mainly the ability to possible gloss, convincingly, as a human being would. Some chatbots make usage also of language tricks, such as monologues, not repeating itself, identify gibberish, play knock-knock jokes, etc. But despite such features, these chatbots can hardly aspire to be somehow *useful* for the task of decision-making, but just to fool their interlocutor. This is because of the historical approach of such a technology, which always aimed, before all, at passing the well-known Turing Test [7] as intelligence evaluation

[3]http://www.github.com/fabiuslongo/ad-caspar
[4]https://www.ocf.berkeley.edu/~arihuang/academic/research/loebner.html

criterion, event though the reader can find copious literature on this theme about which it is insufficient in such a task.

The first distinction between the chatbot platforms divides them into two big macro-categories: *goal-oriented* and *conversational*. The former is the most frequent kind, often designed for business platforms support, assisting users on tasks like buying goods or execute commands in domotic environments. In this case, it is crucial to extract from an utterance the intentions together with all related parameters, then to execute the wanted operation, providing also a proper feedback to the user. As for conversational ones, they are mainly focused on having a conversation, giving the user the feeling to communicate with a sentient being, returning back reasonable answers optionally taking into account discussions topics and past interactions. One of the most common platforms for building conversational chatbot is AIML[5] (Artificial Intelligence Markup Language), based on words pattern-matching defined at design-time; in the last decade it has become a standard for its flexibility to create conversation. In [8], AIML and Chatscript[6] are compared and mentioned as the two widespread opensource frameworks for building chatbots. On the other hand, AIML chatbots are difficult to scale if patterns are manually built, they have great limitations on information extraction capabilities and they are not suitable for task oriented chatbots. Other kinds of chatbots are based on deep learning techniques [9], making usage of huge corpus of examples of conversations to train a generative model that, given an input, is able to generate answers with arguable levels of accuracy.

In general, all chatbots are not easily scalable without writing additional code or repeating the training of a model with fresh datasets. As for the latter, unfortunately neural networks (in particular, the ones used in deep learning applications) suffer from the problem known as *catastrophic interference*: a process where new knowledge overwrites, rather than integrates, previous knowledge. At this regard, the usage of neural models working at semantic tier as dependency parsers, in order to build logical models of utterances in natural language, prevents much more such a drawback.

3. The Architecture

As outlined in the introduction, the proposed architecture, namely AD-Caspar, derives directly from its predecessor Caspar, but, differently from the latter, has been endowed with important features which permit to handle better large KBs and achieve abduction at the same time. The KB is divided into two distinct groups operating separately (orange boxes in Figure 1), which we will distinguish as *Beliefs KB* and *Clauses KB*: the former is managed by the *Belief-Desire-Intention* framework *Phidias* [10], and contains beliefs which support all framework operations, even those related with interaction with environment; the latter contains conceptual information on which we want the agent to make logical inference. Moreover, the Clauses KB is splitted into two different layers: *Low Clauses KB* and *High Clauses KB*. The whole knowledge is stored in the low layer, but the logical inference is achieved in the high one, whose clauses will be the most relevant for the query in exam, taking into account a specific confidence threshold which will be discussed ahead. The two KBs can be seen, somehow, as the two types of human

[5]http://www.aiml.foundation/
[6]https://github.com/ChatScript/ChatScript

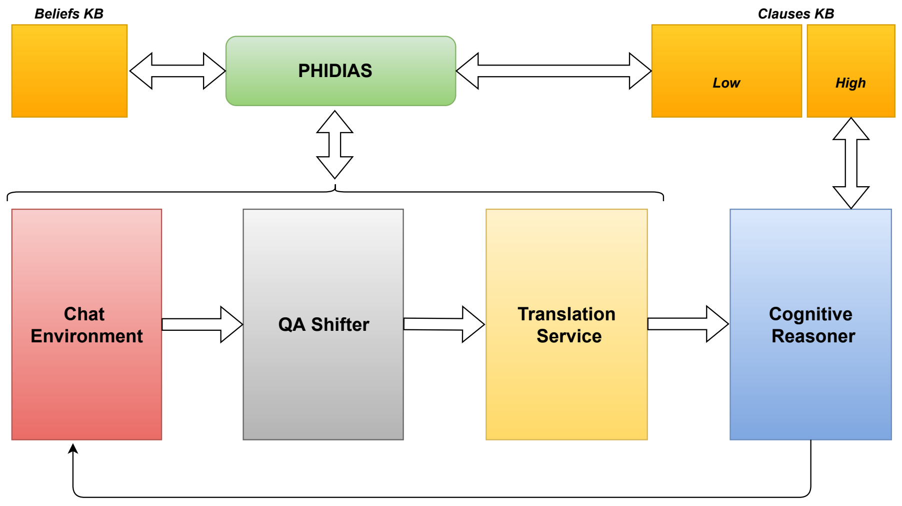


Figure 1: A sempliflied functional scheme of AD-Caspar.

being memory: the so called *procedural memory* or *implicit memory* [11], made of thoughts directly linked to concrete and physical entities; the *conceptual memory*, based on cognitive processes of comparative evaluation. Nevertheless, the two layers of the Clauses KB can be seen as *Short-Term Memory* (High Clauses KB) and *Long-Term Memory* (Low Clauses KB). As well as in human being, in this architecture, Beliefs KB and Clauses KB can interact with each other in a reactive decision-making process (meta-reasoning).

A simplified functional scheme of AD-Caspar is depicted in Figure 1, with each component highlighted with a distinct colour. Specifically, the *Chat Environment* (red box in Figure 1) is responsible of interfacing the agent with all required to establish a communication with a chatbot system. When a question is detected, the text is sent to another module called *QA Shifter* (gray box in Figure 1), whose details are reported in Section 4, with the task of transforming a question into one or more likely assertions as possible answers to such a question. The *Translation Service* (yellow box in Figure 1) is the component responsible of translating a natural language sentence, by leveraging a dependency parser [12] and a production rules system, into in a *neo-Davidsonian* FOL expression. The latter inherits the shape from the event-based formal representation of Davidson [13], with every term t as follows:

$$\texttt{t:= x | L:POS(t}_1\texttt{) | L:POS(t}_1\texttt{, t}_2\texttt{, t}_3\texttt{) | L:POS(t}_1\texttt{, t}_2\texttt{),}$$

where x is a variable bound either to a universal or to an existential quantifier, L a WordNet [14] Synset (selected with a disambiguation process taking in account of the context) or a lemma, POS a Part-of-Speech (POS) tag, and t_1, t_2, t_3 other terms (recursively defined). Implication symbols are also admitted, when a group of predicates subordinate the remaining ones. The outcome of the Translation Service, for a basic verbal phrase, will be as follows:

$$\texttt{synset}_1\texttt{:POS}_1\texttt{(e}_1\texttt{, x}_1\texttt{, x}_2\texttt{)} \wedge \texttt{synset}_2\texttt{:POS}_2\texttt{(x}_1\texttt{)} \wedge \texttt{synset}_3\texttt{:POS}_3\texttt{(x}_2\texttt{),}$$

where $\texttt{synset}_1$ and $\texttt{POS}_1$ are related to a verb, while $\texttt{synset}_2$, $\texttt{POS}_2$, $\texttt{synset}_3$, $\texttt{POS}_3$ to nouns. Additional possible predicates corresponding to other grammatical elements (adjectives, adverbs and prepositions), will share the same variable of predicates which are referred to. Before being sent to the next module, the above formula will be furtherly translated as follows[7], by suppressing the first argument e_1 from the predicate with label $\texttt{synset}_1\texttt{:POS}_1$:

$$\texttt{synset}_1\texttt{:POS}_1\texttt{(synset}_2\texttt{:POS}_2\texttt{(x}_1\texttt{), synset}_3\texttt{:POS}_3\texttt{(x}_2\texttt{)).}$$

The rationale behind such a notation choice is explained next: a definite clause is either atomic or an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. Because of such restrictions, in order to make clauses suitable for inference with the Backward-Chaining algorithm (which requires a KB made of definite clauses), we must be able to encapsulate all their information properly. The strategy followed is to create composite terms, taking into account of the POS and applying the following hierarchy to every noun expression as follows:

$$\texttt{IN(JJ(NN(NNP(x))), t),} \tag{1}$$

where IN is a preposition label, JJ an adjective label, NP and NNP are noun and proper noun labels, x is a bound variable and t a predicate.
As for the verbal phrases, the nesting hierarchy will be the following:

$$\texttt{ADV(IN(VB(t}_1\texttt{, t}_2\texttt{), t}_3\texttt{)),}$$

where ADV is an adverb label, IN a preposition label, VB a verb label, and t_1, t_2, t_3 are predicates; the nesting hierarchy of ADV and IN can also be swapped; in the case of imperative or intransitive verb, instead of respectively t_1 or t_2, the arguments of VB will be left void. As we can see, a preposition (IN) might be related either to a noun or a verb. For a detailed description of the Translation Service, since such component is utterly inherited from CASPAR, the reader is referred to [6]. The *Cognitive Reasoner* (blue box in Figure 1) enables the architecture with reasoning and meta-reasoning capabilities. Differently from the correspondent component in CASPAR, each assertion is made on both High and Low Clauses KB. Since the High Clauses KB is a volatile memory, it will be emptied after the agent is restarted, but at the same time is the one where first the deduction is attempted; then, after a possible unsuccessful reasoning, the Low Clauses KB will be used to populate the High one with fresh nested definite clauses, having a specific number of features compliant with a wanted threshold described ahead in Section 4. After the High Clauses KB is being populated, the deductive reasoning is re-attempted.

4. Question Answering

This section shows how the proposed architecture is able to deal with question-answering. The Low Clauses KB has a very important role in such a task, because it supports abduction as pre-stage of deduction and it is the source from which the High Clauses KB is populated, in order to make logical inference. Each record in the Low Clauses KB is stored in a NoSQL

[7]After such a step the POS can be either removed from the formula as in Figure 6 and Figure 7, or not to keep a shallow label tipization as in Figure 4.

database and it is made of three fields related to a specific sentence in natural language. Each field is defined as follows:

- **Nested Definite Clause**: definite clause made possibly of composite predicates.
- **Features Vector**: vector of labels of the above definite clause.
- **Sentence**: the sentence in natural language.

For instance, let the sentence to be stored be:

Barack Obama became president of United States in 2009.

In this case, the record in the Low Clauses KB will be as follows:

```
- In(Become(Barack_Obama(x1), Of(President(x2), United_States(x3))), 2009(x4))
- [In, Become, Barack_Obama, Of, President, United_States, 2009]
- Barack Obama became president of United States in 2009
```

The abductive strategy of getting useful clauses from the Low Clauses KB, takes into account a metric *Confidence*$_c$ defined as follows:

$$Confidence_c = \frac{|\bigcap(Feats_q, Feats_c)|}{|Feats_q|} \tag{2}$$

where *Feats*$_q$ is the features vector extracted from the query, and *Feats*$_c$ is the features vector of a generic record stored in the Low Clauses KB. A typical access to the Low Clauses KB creates the sorted list of all features occurrences, together with the related clauses, then the most relevant clauses will be copied in the High Clauses KB.

Algorithm 1 aggregate_clauses_greedy(*clause*, *aggregated*, *threshold*)

Input: (i) *clause*: a `definite clause`, (ii) *aggregated*: a set of `definite clauses` (empty in the first call), (iii) *threshold*: the minimum `confidence` threshold
Output: a set of `definite clauses`.

```
 1: ft ← extract_features(clause);
 2: aggr ← get_relevant_clauses_from_db(ft);
 3: foreach record in aggr do
 4:     occurrencies_found ← record.features_occurrencies
 5:     confidence ← occurrencies_found / size(ft)
 6:     foreach cls in record.clauses do
 7:         if cls not in aggregated and confidence ≥ threshold then do
 8:             aggregated.append(cls);
 9:             aggregate_clauses_greedy(cls, aggregated, threshold);
10: return aggregated
```

Such an operation is accomplished via the `aggregate_clauses_greedy` algorithm shown in Algorithm 1. The latter, with a greedy heuristic, takes in input the query *clause*, the set *aggregated* and the wanted limitation of confidence *threshold* for abduction; as output, it gives back the set *aggregated* of clauses from the db that are going to be copied in the High Clauses KB, taking in account of the wanted distance (2) from the query.

Table 1
A simple instance of clauses related features.

Clauses	Features
$\texttt{cls}_1$	`[a, x, z, y]`
$\texttt{cls}_2$	`[a, b]`
$\texttt{cls}_3$	`[a, b, x, y]`
$\texttt{cls}_4$	`[a, b, c, d]`
$\texttt{cls}_5$	`[a, b, c, w]`

4.1. Algorithm

First, at line 1 of Algorithm 1, `aggregate_clauses_greedy` extracts all *clause* features; at line 2, it creates a list of tuples in descending order (by the first field) containing an integer value and a lists of clauses having in common such value as number of common features with *clause*. More in detail, considering the Table 1, having a query $\texttt{cls}_q$ with features vector `[a, b, c]`, the function `get_relevant_clauses_from_db` will create a list made of the following two tuples, by excluding those with only one feature in common, since in such a domain a minimal meaningful verbal phrase[8] is made at least of two entities (verb plus subject):

$$\texttt{(3, [cls}_4\texttt{, cls}_5\texttt{])}$$
$$\texttt{(2, [cls}_2\texttt{, cls}_3\texttt{])}$$

The rationale behind such function's output is that the first value of each tuple is the size of the intersection between the features of the query and the features of Table 1 (*feature_occurencies* in Algorithm 1), while the second value (*clauses* in Algorithm 1) is a vector containing the clauses themselves having in common such an intersection size. At line 4, the first value of each tuple is extracted and used in line 5 to calculate the confidence (2). In the loop at line 6, all clauses having the same features occurrences are considered, and at line 7 the algorithm checks whether the clause has an admitted confidence level and it is not already in *aggregated*. In this case, the clause will be appended to the *aggregated* list. At line 9, there is a recursive call taking in input *cls* (the current clause which is being processed) instead of *clause*, the updated *aggregated* and the *threshold* of the same procedure call. Finally, at line 10, the list *aggregated* is returned.

Although there can be more strategies to implement the function at line 2 of Algorithm 1, for this work's prototype the desired result has been achieved by leveraging the Mongodb aggregation operator, in a single fast and efficient database operation shown in Figure 2. The latter is a pipeline of four different operations: the first, at line 4, is the processing of all possible sizes of intersections between features fields in the db and the features of the query clause. At line 6 all clauses with the common value of such a size are grouped in tuples. At line 7 such tuples are sorted by the intersection size. Finally, at line 8 the output is limited to the two most significant tuples. Nonetheless, we can affirm the algorithm is sound, accomplishing its job in at most:

$$O(t * s^2)$$

[8]For instance a reflective verbal phrase as: *Roy runs*.

```
aggr = db.clauses.aggregate([
      {"$project": {
          "value": 1,
          "intersection": {"$size": {"$setIntersection": ["$features", features]}}
      }},
      {"$group": {"_id": "$intersection", "group": {"$push": "$value"}}},
      {"$sort": {"_id": -1}},
      {"$limit": 2}
   ])
```

Figure 2: The Python Mongodb aggregate operator implementing the function `get_relevant_clauses_from_db` at line 2 of Algorithm 1.

with $0 < t < 1$ as the confidence threshold and s the size of the Low Clauses KB.

4.2. Polar Questions

Polar questions in the shape of nominal assertion (excepting for the question mark at the end) are transformed into nested definite clauses and treated as query as they are, while those beginning with an auxiliary term, for instance:

Has Margot said the truth about her life?

which can be distinguished by means of the following dependency:

`aux(said, Has)`

will be treated by removing the auxiliary and considering the remaining text (without the ending question mark) as source to be converted into a clause-query.

4.3. Wh-Questions

Differently from polar questions, for dealing with wh-question we have to transform a question into one or more assertions that can be expected as likely answer, then to query the agent with them. To achieve that, after an analysis of several types of questions for each category[9], by leveraging the dependencies of the questions, we found it useful to divide the sentences text into specific chunks as it follows:

`[PRE_AUX][AUX][POST_AUX][ROOT][POST_ROOT][COMPL_ROOT]`

The delimiter indexes between every chunk are given starting from the AUX and ROOT dependencies positions in the sentence. The remaining chunks are extracted on the basis of the latters. For the likely answers composition, the module *QA Shifter* has the task of recombining the question chunks in a different permutation, depending on the idiom in exam, considering also the wh-question type. Such an operation, which is strictly language specific, is accomplished thanks to an *ad-hoc* production rule system which takes in account of languages diversity. For instance, let the question in exam be:

[9]Who, What, Where, When, How.

Who could be Biden?

In this case, the chunks sequence will be as follows:

```
[PRE_AUX][could][POST_AUX][be][Biden][COMPL_ROOT]
```

where only `AUX`, `ROOT` and `POST_ROOT` are populated, while the other chunks are empty. In this case a specific production rule of the QA Shifter will recombine all chunks in a different sequence, by adding also another specific word in order to compose a likely assertion as follows:

```
[PRE_AUX][POST_AUX][Biden][could][be][COMPL_ROOT][Dummy]
```

At this point, joining all the words in such a sequence, the candidate sentence as likely assertion to used as query, might be the following one:

Biden could be Dummy.

The meaning of the keyword *Dummy* will be clarified shortly. In all verbal phrases where `ROOT` is a copular[10] verb (like *be*), the verb has the same properties of the identity function. Then, in the case of the above sentence, we can consider also the following candidate sentence:

Dummy could be Biden.

All wh-questions for their own nature require a *factoid* answer, made of one or more words (snippet); so, in the presence of the question: *Who is Biden?* as answer we expect something like:

Biden is something. (3)

But *something* surely is not what we are looking for as information, but *the elected president of United States* or something similar. This means that, within the FOL expression of the query, *something* in (3) must be represented by a mere variable. In light of this, instead of *something*, this architecture uses the keyword *Dummy*: the rationale of this choice is that, during the creation of a FOL expression containing such a word, the Translation Service will impose the extra POS `DM` to *Dummy*, whose parsing is not expected to be used to build a nested definite clause, thus it will be discarded. At the end of this process, as FOL expression of the query we'll have the following literal:

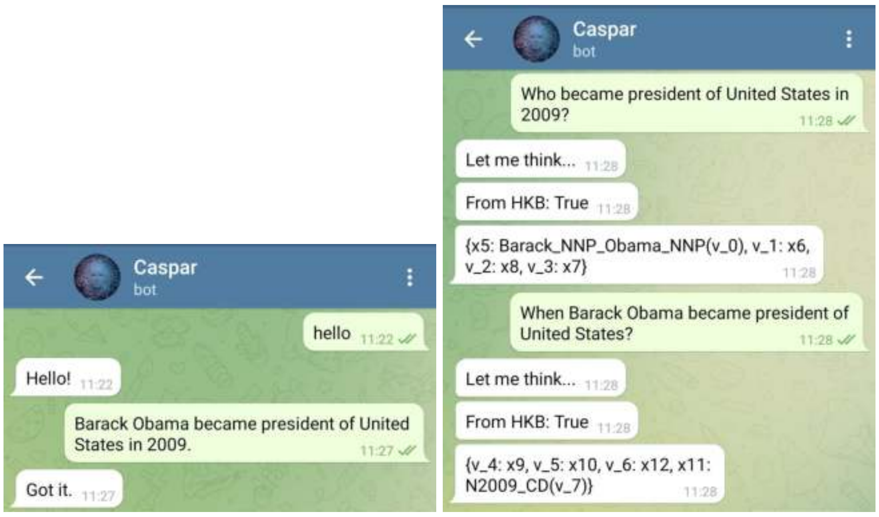
`Be(Biden(x1), x2),` (4)

which means that, if the High Clauses KB contains the representation of *Biden is the president of America*, namely:

```
Be(Biden(x1), Of(President(x2), America(x3))),
```

querying with the (4) by using the Backward-Chaining algorithm, the latter will return back as result a unifying substitution with the previous clause as follows:

[10]A copular verb is a special kind of verb used to join an adjective or noun complement to a subject. Common examples are: be (is, am, are, was, were), appear, seem, look, sound, smell, taste, feel, become, and get. A copular verb expresses either that the subject and its complement denote the same thing or that the subject has the property denoted by its complement.



hello 11:22

Hello! 11:22

Barack Obama became president of United States in 2009. 11:27

Got it. 11:27

Who became president of United States in 2009? 11:28

Let me think... 11:28

From HKB: True 11:28

{x5: Barack_NNP_Obama_NNP(v_0), v_1: x6, v_2: x8, v_3: x7} 11:28

When Barack Obama became president of United States? 11:28

Let me think... 11:28

From HKB: True 11:28

{v_4: x9, v_5: x10, v_6: x12, x11: N2009_CD(v_7)} 11:28

Figure 3: Starting a Telegram chat session with an instance of AD-Caspar (left) and querying it with *who* and *when* questions (right).

```
{v_41: x1, x2: Of(President(v_42), America(v_43))},
```

which contains, in correspondence of the variable x2, the logic representation of the snippet: *president of America* as possible and correct answer. Furthermore, starting from the lemmas composing the only answer-literal within the substitution, with a simple operation on a string, it is possible to extract the minimum snippet from the original sentence containing such lemmas.

5. Case-study

In this section, a simple Python prototype of a Telegram chatbot based on the AD-Caspar architecture is shown, focusing on how it deals with each type of interaction with the user.

5.1. Starting, Asserting and Querying the chatbot

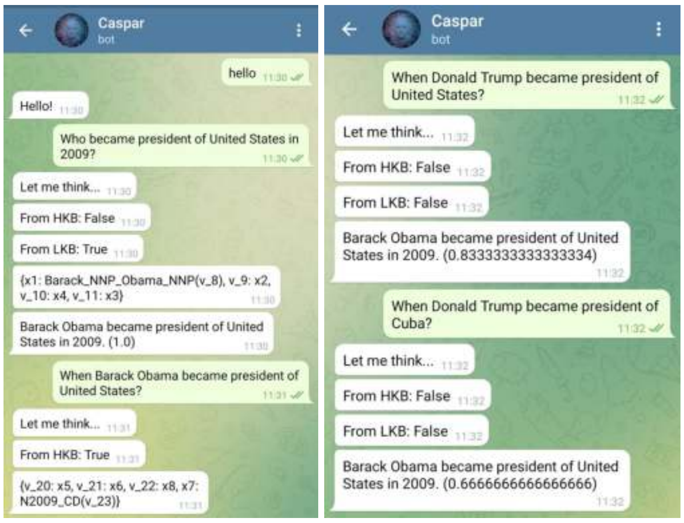
As the agent is running, to start a new session the user has to provide the keyword *hello*, to make the agent come out from its idle state, then feed or enquiry the chatbot with the wanted information.

In Figure 4 it is shown the content of both High and Low Clauses KB in the case of new assertions, after the events of Figure 3 (left), by means the AD-Caspar native commands `hkb()` and `lkb()`. In Figure 3 (right) we can see how the chatbot is queried with *wh-questions*, giving back as result a substitution from the High Clauses KB (`From HKB: True`) containing a literal, which is a logic representation of a snippet-result in natural language. Regarding the substitution of the variable x5, e.g., the literal is the representation of the snippet *Barack Obama*, whose words are concatenated together their POS.

After the chatbot reboot, as we can see for instance in Figure 5 (left), the result is extracted from Low Clauses KB (`From LKB: True`) taking into account the confidence threshold (0.6

```

eShell: main > hkb()

In_IN(Become_VBD(Barack_NNP_Obama_NNP(x1), Of_IN(President_NN(x2), United_NNP_States_NNP(x3))), N2009_CD(x4))

1 clauses in High Knowledge Base

eShell: main > lkb()

In_IN(Become_VBD(Barack_NNP_Obama_NNP(x1), Of_IN(President_NN(x2), United_NNP_States_NNP(x3))), N2009_CD(x4))
['In_IN', 'Become_VBD', 'Barack_NNP_Obama_NNP', 'Of_IN', 'President_NN', 'United_NNP_States_NNP', 'N2009_CD']
Barack Obama became the president of United States in 2009.

1  clauses in Low Knowledge Base
```

Figure 4: AD-Caspar High and Low Clauses KB changes, after assertions.



hello 11:30
Hello! 11:30
Who became president of United States in 2009? 11:30
Let me think... 11:30
From HKB: False 11:30
From LKB: True 11:30
{x1: Barack_NNP_Obama_NNP(v_8), v_9: x2, v_10: x4, v_11: x3} 11:30
Barack Obama became president of United States in 2009. (1.0) 11:30
When Barack Obama became president of United States? 11:31
Let me think... 11:31
From HKB: True 11:31
{v_20: x5, v_21: x6, v_22: x8, x7: N2009_CD(v_23)} 11:31

When Donald Trump became president of United States? 11:32
Let me think... 11:32
From HKB: False 11:32
From LKB: False 11:32
Barack Obama became president of United States in 2009. (0.8333333333333334) 11:32
When Donald Trump became president of Cuba? 11:32
Let me think... 11:32
From HKB: False 11:32
From LKB: False 11:32
Barack Obama became president of United States in 2009. (0.6666666666666666) 11:32

Figure 5: On the left, querying the chatbot with *who* and *when* questions, after the rebooting and the Low Clauses KB was fed. On the right, abductive results with confidence threshold equal to 0.6, after querying with *when* questions.

in this case), because High Clauses KB is still empty (`From HKB: False`). Such a threshold, depending on the domain, can be changed by modifying the value of a specific parameter. In the bot closed-world assumption, the agent can give back only answers unifying with the content of its own knowledge, otherwise it will return *False*. Optionally, by setting another parameter in a configuration file, the closest results can be shown together with their confidence (Figure 5 right).

5.2. Runtime Evaluation

In the scope of chatbot applications, the issue of responsiveness is worth of attention, because the user should have the feeling, somehow, of relating to a sentient being providing reasonable

```
Be(Nono(x1), Hostile(Nation(x2)))
Be(Colonel_West(x1), American(x2))
Be(Missile(x1), Weapon(x2))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3))
To(Sell(American(x1), Weapon(x2)), Hostile(Nation(x3))) ==> Be(American(x4), Criminal(x5))
```

Figure 6: AD-Caspar High Clauses after five assertions related to the *Colonel West* KB.

response times. In light of above, to address such an issue, since a chatbot relays on the internet, its real-time performances depends firstly on the bandwidth and secondly on the chatbot engine. Since the bandwidth is the more volatile parameter between the two, because it depends on physical features, protocols and temporally congestion, in this work we decided to focus only on the engine performances considering a widespread specific hardware. For this reason, in order to achieve a runtime evaluation of the engine, an instance of AD-Caspar was tested, whose timings in seconds are shown in Table 2. The hardware the chatbot has been tested on is the Intel i7-8550U 1.80Ghz with 8GB RAM. The first column `Knowledge Base` of Table 2 refers to three distinct KBs, each with different size, containing the clauses in Figure 7. Such clauses are asserted starting from the five sentences related to the *Colonel West* case, namely:

- *Colonel West is American.*
- *Nono is a hostile nation.*
- *missiles are weapons.*
- *Colonel West sells missiles to Nono.*
- *When an American sells weapons to a hostile nation, that American is a criminal.*

whose corresponding FOL notation, seen before in Section 3 (see also Figure 6), is not sufficient to infer that *Colonel West is a criminal.*
For this reason, together with each sentence related clause, AD-Caspar adopts a specific *expansion* of the KB, inherited from Caspar, which leverages the so-defined *assignment rules* and *clause conceptual generalizations.* For the sake of shortness we will not report the details of such an expansion, for which the reader is referred to [6]. Anyway, such an expansion improves the chances of successful reasoning, by increasing the clauses from 5 in Figure 6 up to 25 in Figure 7. The first column in Table 2 is about three distinct KBs: the first, namely *West25*, contains exactly such 25 clauses; *West104* and *West303* contain either such clauses, but respectively plus 79 and plus 278 random unrelated clauses. The column HKB of Table 2 refers to five computation timings for each of the KBs, considering only the High Clauses KB and the query: *Colonel West is a criminal.* We remind that an instance of AD-Caspar attempts firstly to achieve a reasoning making usage of only the High Clauses KB, as in the case of Caspar, otherwise it will get likely candidates from the Low Clauses KB, considering their relevance to the query according to a specific threshold confidence (2). The third column LKB+HKB of Table 2 shows timings in the case the High Clauses KB is initially empty, thus both High Clauses KB and Low Clauses KB are involved. Focusing on the third column, it appear clear timings in general are lesser than the second column, excepting for the values in first rows, which comprises the Mongodb access time and the filling of the High Clauses KB with clauses coming

```
Be(Nono(x1), Nation(x2))
Be(Nono(x1), Hostile(Nation(x2)))
Nono(x) ==> Nation(x)
Nono(x) ==> Hostile(Nation(x))
Be(Colonel_West(x1), American(x2))
Colonel_West(x) ==> American(x))
Be(Missile(x1), Weapon(x2))
Missile(x) ==> Weapon(x)
Sell(Colonel_West(x1), Missile(x2)) ==> Sell(American(v_0), Missile(x4))
Sell(Colonel_West(x1), Missile(x2)) ==> Sell(American(x3), Weapon(v_1))
Sell(Colonel_West(x1), Missile(x2)) ==> Sell(Colonel_West(x1), Weapon(v_2))
Sell(Colonel_West(x1), Missile(x2))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(x1), Missile(x2)), Nation(v_4))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_5), Missile(v_6)), Nation(v_4))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_7), Weapon(v_8)), Nation(v_4))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(v_9), Weapon(v_10)), Nation(v_4))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(x1), Missile(x2)), Hostile(Nation(v_11))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_12), Missile(v_13)), Hostile(Nation(v_11))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_14), Weapon(v_15)), Hostile(Nation(v_11))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(v_16), Weapon(v_17)), Hostile(Nation(v_11))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_18), Missile(v_19)), Nono(x3))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(American(v_22), Weapon(v_23)), Nono(x3))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3)) ==> To(Sell(Colonel_West(v_26), Weapon(v_27)), Nono(x3))
To(Sell(Colonel_West(x1), Missile(x2)), Nono(x3))
To(Sell(American(x1), Weapon(x2)), Hostile(Nation(x3))) ==> Be(American(x4), Criminal(x5))
```

Figure 7: AD-Caspar *Colonel West* KB in Figure 6 after its expansion.

from the Low one, via the `aggregate_clauses_greedy` in Algorithm 1. The other values in the third column are lower than in the second one because the reasoning is achieved over a lesser number of clauses (19) for each distinct KB. The average timings in the bottom row show how the first rows' value is amortized, by compensating the loss, due to the gain achieved from reasoning on a fewer number of clauses than respect the initial content of all KBs in exam (*West25*, *West104* and *West303*). Intuitively it is expected such bias to be increased for larger KBs, which demonstrates the effectiveness of such approach for two distinct tasks: firstly, to deal with larger KBs considering only the most relevant clauses in the reasoning process; secondly, to permit at the same time abduction as pre-stage of deduction, in order to give back closer results also in presence of non-successful reasoning.

6. Conclusions and Future works

In this paper, a framework based on natural language processing and first-order logic, with the aim of instantiating *cognitive* chatbots able of abductive-deductive reasoning, was presented. By the means of its module Translation Service, AD-Caspar parses sentences in natural language in order to populate its KBs with beliefs or *nested* definite clauses endowed of rich semantic. Moreover, the module QA Shifter is able to reshape wh-questions into likely assertions one can expect as answer, thanks to a production rule system leveraging a dependency parser. The combination of Translation Service and QA Shifter makes the Telegram Bot proposed in this work easily scalable on the knowledge we want it to deal with, because the user has to provide

Table 2
Real-time cognitive performances (in seconds) of a Telegram chatbot engine based on AD-Caspar, in the case of successful reasoning with KBs of different sizes.

Knowledge Base	HKB	LKB+HKB
	0,377	0,469
West25	0,378	0,353 (19/25)
	0,437	0,385 (19/25)
	0,374	0,366 (19/25)
	0,355	0,399 (19/25)
	0,423	0,426
West104	0,362	0,327 (19/104)
	0,342	0,327 (19/104)
	0,353	0,388 (19/104)
	0,731	0,323 (19/104)
	0,407	0,463
West303	0,421	0,357 (19/303)
	0,377	0,333 (19/303)
	0,461	0,368 (19/303)
	0,443	0,387 (19/303)
Overall AVG	0,416	0,378

only the new sentences in natural language at runtime, like in a common conversation. As future work, we plan to include a module for the design of enhanced dialog systems, taking in account of contexts and history, and also integrate a module for Argumentation.

References

[1] M. Minsky, A framework for representing knowledge. In P. Winston, Ed., *The Psychology of Computer Vision*, New York: McGraw-Hill, 1975.

[2] . A. Schank, R., Scripts, plans, goals, and understanding: An inquiry into human knowledge structures, Hillsdale, NJ: Erlbaum, 1977, pp. 211–277.

[3] C. F. Longo, C. Santoro, F. F. Santoro, Meaning Extraction in a Domotic Assistant Agent Interacting by means of Natural Language, in: 28th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE, 2019.

[4] C. S. Carmelo Fabio Longo, Ad-caspar: Abductive-deductive cognitive architecture based on natural language and first order logic reasoning, in: 4th Workshop on Natural Language for Artificial Intelligence (NL4AI 2020) co-located with the 19th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2020), 2020.

[5] J. H. Kotagiri Ramamohanarao, An introduction to deductive database languages and systems, The International Journal of Very Large Data Bases Journal, 3, 107-122 (1994).

[6] C. F. Longo, F. Longo, C. Santoro, Caspar: Towards decision making helpers agents for iot, based on natural language and first order logic reasoning, Engineering Applications of Artificial Intelligence 104 (2021) 104269. URL: https://www.sciencedirect.com/science/article/pii/S0952197621001160. doi:https://doi.org/10.1016/j.engappai.2021.104269.

[7] A. Turing, Computing machinery and intelligence, Mind, 1950, pp. 433–60.
[8] H. Madhumitha.S, Keerthana.B, Interactive chatbot using aiml, Int. Jnl. Of Advanced Networking & Applications Special Issue (2019).
[9] Q. V. L. Ilya Sutskever, Oriol Vinyals, Sequence to sequence learning with neural networks, Advances in Neural Information Processing Systems 27 (2014).
[10] C. S. Fabio D'Urso, Carmelo Fabio Longo, Programming intelligent iot systems with a python-based declarative tool, in: The Workshops of the 18th International Conference of the Italian Association for Artificial Intelligence, 2019.
[11] D. Schacter, Implicit memory: history and current status, Journal of Experimental Psychology: Learning, Memory, and Cognition vol. 13, 1987, pp. 501–518 (1987).
[12] A. S. Jinho D. Choi, Joel Tetreault, It depends: Dependency parser comparison using a web-based evaluation tool, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, 2015, p. 387–396.
[13] D. Davidson, The logical form of action sentences, in: The logic of decision and action, University of Pittsburg Press, 1967, p. 81–95.
[14] G. A. Miller, Wordnet: A lexical database for english, in: Communications of the ACM Vol. 38, No. 11: 39-41, 1995.

Abduction in (Probabilistic) Answer Set Programming

Damiano Azzolini[1], Elena Bellodi[2] and Fabrizio Riguzzi[3]

[1]*Dipartimento di Scienze dell'Ambiente e della Prevenzione, University of Ferrara, Ferrara, Italy*
[2]*Dipartimento di Ingegneria, University of Ferrara, Ferrara, Italy*
[3]*Dipartimento di Matematica e Informatica, University of Ferrara, Ferrara, Italy*

Abstract
Answer Set Programming (ASP) is a branch of Logic Programming particularly useful for representing complex domains. Logic abduction, the reasoning strategy that deals with incomplete data, is tightly related to ASP, and encodes incompleteness through abducibles. The goal of logic abduction is to find the minimal set of abducibles (where minimality is usually considered in terms of set inclusion) that explains a query. Recently, abductive reasoning has been introduced in the context of Probabilistic Logic Programming, but no solutions are available for Probabilistic Answer Set Programming (PASP). In this paper, we close this gap and propose an algorithm to perform abduction both in ASP and in PASP.

Keywords
Abduction, Statistical Relational Artificial Intelligence, Probabilistic Answer Set Programming

1. Introduction

Abductive Logic Programming (ALP) [1, 2] is an extension of Logic Programming (LP) [3] that copes with incomplete data: given a set of *abducible* facts and a set of *integrity constraints*, the goal is to find the *minimal* set, where minimality is usually considered in terms of set inclusion, that explains a given query. This minimal set is often called *abductive explanation*.

Answer Set Programming (ASP) [4] is a powerful formalism to encode complex problems, where the possible solutions are represented as *answer sets*. In the first contribution of this paper, we propose a simple yet effective algorithm to perform abduction in ASP.

One limitation of ASP (and Logic Programming in general) is that it cannot manage uncertain data. Probabilistic Logic Programming (PLP) [5, 6] under the Distribution Semantics (DS) [7] is a possible formalism to express uncertain information using a logic-based language. Recently, the authors of [8] introduced the concept of *probabilistic abductive explanation* and proposed an algorithm to perform abduction in PLP under the DS. A possible extension to ASP that manages uncertainty is *Probabilistic* Answer Set Programming (PASP) under the Credal Semantics (CS) [9, 10]. With this semantics, the probability of a query is not a sharp value, but it is represented by an interval, i.e., it has a lower and an upper probability bound. In a second contribution of this paper, we propose an algorithm to perform abduction in PASP, where the goal is to find, given a query, the minimal set of abducible facts that maximizes the joint *lower probability* of the query and the integrity constraints.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
damiano.azzolini@unife.it (D. Azzolini); elena.bellodi@unife.it (E. Bellodi); fabrizio.riguzzi@unife.it (F. Riguzzi)

Overall, our contribution is two-fold: first, we provide an algorithm to compute abductive explanations in ASP. Then, we introduce the concept of abductive reasoning in PASP under the CS and propose an algorithm to compute probabilistic abductive explanations. To study the applicability of our approach, we tested both algorithms on three different datasets. The results show that abduction for ASP is orders of magnitude faster than for PASP.

The paper is structured as follows: Section 2 introduces the basic concepts of ASP, PLP, and PASP under the CS. Section 3 describes our algorithm to perform abduction in ASP while our proposal for PASP under the Credal Semantics is presented in 4. The two algorithms are tested in Section 5, related work is discussed in Section 6, and Section 7 concludes the paper.

2. Background

2.1. Answer Set Programming

We assume the reader is already familiar with the basic notions of Logic Programming. For an in-depth treatment of the subject, see [3]. ASP also considers aggregate atoms [11] of the form $g_0 \circ_0 \#f\{e_1; ...; e_n\} \circ_1 g_1$ where g_0 and g_1 are constants or variables and are called *guards*, f is an aggregate function symbol, and $\circ_0$ and $\circ_1$ are arithmetic comparison operators. Each e_i is an expression of the form $t_1, ..., t_l : C$ where each t_i is a term whose variables appear in C, a conjunction of literals. For example, a possible aggregate is $1 < \#count\{X : f(X)\} < 5$. $g_0 \circ_0$ or $\circ_1 g_1$ or both may be omitted.

A rule is of the form $h_1; ...; h_n \leftarrow b_1, ..., b_m$, where each h_i is an atom and each b_i is a literal. $h_1; ...; h_n$ is called the *head* and $b_1, ..., b_m$ is called the *body*. We only consider *safe rules*, i.e., rules where each variable of a rule also appears in a positive literal in the body. If $n = 0$ (no atoms in the head) and $m > 0$, the rule is called an *integrity constraint*. If $n = 1$ and $m = 0$, the rule is called a *fact*, and represents what is known to hold. If a rule does not contain variables, it is called *ground*. The set of groundings of a rule can be obtained by replacing its variables with the constants that appear in the program in all possible ways. An answer set program is a set of rules.

The Herbrand base (B_P) of an answer set program P is the set of all ground atoms that can be constructed using the symbols in the program. An *interpretation* I for P is a subset of B_P. I satisfies a ground rule if at least one h_i is true in I when every b_i is true in I. A *model* is an interpretation that satisfies all the groundings of all the rules of P. If we consider a ground program P_g and an interpretation I, by removing from P_g the rules in which a b_i is false in I we obtain the *reduct* [12] of P_g with respect to I. An *answer set* for P is an interpretation I such that I is a minimal model (in term of set inclusion) of the reduct of P_g. With $AS(P)$ we denote the set of all the answer sets of P.

2.2. Probabilistic Logic Programming (PLP)

Probabilistic Logic Programming [5, 6] extends Logic Programming by incorporating probabilities into facts or rules. If we consider, for instance, ProbLog [13], a probabilistic fact is represented with $\Pi :: f$ where $\Pi \in]0, 1]$ and f is a logical atom. The Distribution Semantics (DS) [7] is at the heart of many PLP languages, such as ProbLog. Following the DS, an *atomic* *choice* is represented with a tuple (f, θ, k) where $k \in \{0, 1\}$: $k = 1$ means that the grounding $f\theta$ for f is selected; $k = 0$ indicates that it is not. A consistent set of atomic choices is called a *composite choice* (identified with κ) and its probability can be computed as

$$P(\kappa) = \prod_{(f_i,\theta,1)\in\kappa} \Pi_i \cdot \prod_{(f_i,\theta,0)\in\kappa} (1 - \Pi_i)$$

If a composite choice contains an atomic choice for every grounding of every probabilistic fact then it is called a *selection*. A selection identifies a *world*, i.e., a logic program composed by the rules of the program and the selected probabilistic facts (those for which $k = 1$). The probability of a world w, $P(w)$, is the probability of the correspondent selection. Finally, the probability of a *query* q (a conjunction of ground atoms) is computed as

$$P(q) = \sum_{w \models q} P(w)$$

Probabilistic facts are considered independent. For example, the following (propositional) program

```
0.3::nosleep.
0.6::lowvitamins.
tired:- nosleep.
tired:- lowvitamins.
```

has 2 probabilistic facts: `nosleep`, that is true with probability 0.3, and `lowvitamins`, that is true with probability 0.6. The program has $2^2 = 4$ worlds; the query `tired` is true in 3 of them (those where at least one of the two probabilistic facts is true) and it has probability $0.3 \cdot 0.6 + 0.3 \cdot (1 - 0.6) + (1 - 0.3) \cdot 0.6 = 0.72$.

2.3. Probabilistic ASP under the Credal Semantics

The Distribution Semantics only considers probabilistic logic programs where every world has a unique two-valued well-founded model [14]. This usually does not hold if we consider ASP programs with probabilistic facts (PASP).

In this case, the Credal Semantics (CS) [9, 10] has been proposed as a possible underlying semantics. Under this semantics, every query q has lower and upper probability bounds, denoted respectively with $\underline{P}(q)$ and $\overline{P}(q)$. In addition to the worlds, the computation of the probability for a query also requires considering the answer sets for each of them: if the query is true in *at least one* answer set of a world w, $P(w)$ contributes to the upper probability; if the query is true in *every* answer set of a world w, $P(w)$ contributes to the lower probability. Clearly, $\underline{P}(q) \leq \overline{P}(q)$. However, every world must have at least one answer set. If we slightly modify the previous program in

```
0.3::nosleep.
0.6::lowvitamins.
tired:- nosleep.
tired; nottired:- lowvitamins.
```

and still consider the query `tired`, the worlds where both probabilistic facts are true and the worlds where `nosleep` is true and `lowvitamins` is false have only 1 answer set each, {`nosleep, tired, lowvitamins`} and {`nosleep, tired`} respectively, and the query is true in them, so we have a contribution of $0.3 \cdot 0.6 + 0.3 \cdot (1 - 0.6)$ to the lower probability. The world where `lowvitamins` is true and `nosleep` is false has 2 answer sets ({`tired, lowvitamins`} and {`nottired, lowvitamins`}) and the query is true in only one of the two, so we get a contribution of $(1 - 0.3) \cdot 0.6$ to the upper probability. In the world where both probabilistic facts are false, the query is false as well so it does not contribute to the probability. Overall, the probability lies in the range $[0.3, 0.72]$.

3. Abductive Answer Set Programming

An *abductive answer set program* is composed of an answer set program and a set of ground atoms called the *abducibles*, that do not appear in the head of any rule. More formally, given an answer set program P, and a possibly empty set of abducibles (also called abducible facts) A, the goal is to find the minimal set of abducibles Δ, called *abductive explanation*, such that a *query* is present in *at least one* answer set. If there are multiple minimal sets, we call them the abductive explanations. Here, minimality is intended in terms of set inclusion [15]. For example, if both $\Delta^{'}$ = {`a`} and $\Delta^{''}$ = {`a,b`} are explanations for a query q, only $\Delta^{'}$ is considered as the abductive explanation, since $\Delta^{''} \supset \Delta^{'}$, and thus $\Delta^{''}$ is not minimal. We consider integrity constraints (ICs) as normal answer set constraints. To better illustrate these concepts, consider the following example.

Example 1 (Smoke). *The program and the graph of Figure 1 describe a network with 5 people (nodes) connected by a friendship relation (edges). The friendship relation exists if the corresponding abducible (denoted by prepending the functor `abducible` to the terms `e/2`) is selected. Some individuals smoke, some others do not. In particular, `b` and `d` smoke. A disjunctive rule states that a person `X` can either smoke or not smoke given that she is a friend of someone (`Y`) who smokes. The integrity constraint states that at least 80% of the people smoke. The goal is to compute the abductive explanation(s) for the query `smokes(c)`.*

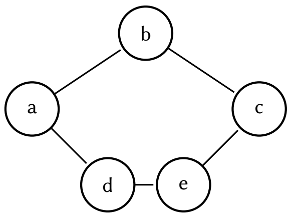
We propose an algorithm to perform abduction in ASP.

Every abducible fact `abducible a` is replaced by a choice rule of the form `0`{`a`}`1`, stating that `a` can be included or not in every answer set. To encode the query `query`, we add a constraint `:- not query`. In this way, we impose that the query is true in every answer set. By applying this transformation to the program shown in Figure 1a, we obtain a new program with 62 answer sets. Every answer set represents a possible explanation but only some of them (2) are abductive explanations, i.e., minimal (in terms of set inclusion), for the query `smokes(c)`: {{`e(b,c)`},{`e(d,e),e(e,c)`}}.

To compute the abductive explanations, a first solution could consist in enumerating all the answer sets and removing the dominated ones. However, the number of abductive explanations is usually orders of magnitude smaller than the possible answer sets (in this example, there are 2 abductive explanations, while the total number of answer sets is 62). For this reason we adopt an alternative approach. First, we compute the cautious consequences (intersection of all the

```
abducible e(a,b). abducible e(b,c).
abducible e(a,d). abducible e(d,e).
abducible e(e,c).

friend(X,Y):- e(X,Y).
friend(X,Y):- e(Y,X).

smokes(b). smokes(d).

smokes(X) ; nosmokes(X):-
    friend(X,Y), smokes(Y).

:- #count{X:nosmokes(X)} = N,
   #count{X:smokes(X)} = S,
   10*S < 8*(N+S).
```

(a) Program.



(b) Network of 5 people (nodes) connected by a friendship relation (edges).

Figure 1: Example of an abductive answer set program and its graph representation.

models) of the whole program, and project [16] them on the abducibles. For every abducible a in the cautious consequences, we add a constraint `:- a` in the program, since these are present in every answer set, so we can avoid generating a choice for them. After this process, we add an additional rule with an argument that stores the number of abducibles selected in the computed answer sets. For example, for the program shown in Example 1a, we add `c(C):- #count{Y,X : e(X,Y)} = C`. After that, we iteratively generate answer sets and, at each step, we add a constraint to the program that imposes that the number of abducibles in the generated answer sets is N, where N ranges from 0 to the total number of abducibles in the program. We go from 0 to N, so we can keep track of the dominated explanations. The answer sets obtained at each iteration are the abductive explanations. In other words, we call multiple times the solver to generate the answer sets and, at each call, we generate only the answer sets with a fixed number of abducibles. Moreover, at each iteration, we also impose a constraint to avoid the generation of answer sets that are supersets of the already computed ones. This process is described in Algorithm 1: first, abducibles are converted as previously described (line 2). Then, we compute the minimal set of abducible facts (line 3) and add each fact of this minimal set into the program, as integrity constraint (line 5). The loop at line 9 handles the generation of sets of abducibles of increasing size. The function AddConstraintSizeAndComputed adds a constraint to the program to limit the number of abducibles and a constraint to remove the already computed solutions. At the end of the loop, we check whether there are some solutions that are not minimal with the function Least and return the abductive explanation for the query.

To clarify the overall process, let us apply Algorithm 1 on the program of Example 1. There are no cautious consequences projected on the abducibles, so no constraint is added to the program. Then, we begin generating the answer sets, starting from 0 abducibles with the constraint `:- c(X), X != 0`. This program is unsatisfiable, so no further constraints are added. In the

Algorithm 1 Function AbductionASP: computation of the abductive explanations for a query *query* and an ASP program $\mathcal{P}$.

```
1: function Abduction($query, \mathcal{P}$)
2:     $probFacts, abducibles, P_p \leftarrow$ ConvertProgram($\mathcal{P}$)
3:     $minSet \leftarrow$ ComputeMinimalSet($P_p \cup \{$:- $not\ query.\}$)
4:     for all $f \in minSet$ do
5:         $P_p \leftarrow P_p \cup \{$:- $not\ a.\}$
6:     end for
7:     $alreadyComputed \leftarrow \emptyset$
8:     $abduciblesSet \leftarrow \emptyset$
9:     for $i \in range(0, len(abducibles))$ do
10:        $P_p^c \leftarrow$ AddConstraintSizeAndComputed($P_p, i, alreadyComputed$)
11:        $P_p^{qc} \leftarrow P_p^c \cup \{$:- $not\ query.\}$
12:        $projectSet \leftarrow abducibles$
13:        $AS \leftarrow$ ProjectSolutions($P_p^{qc}, projectSet$)          ▷ Computation of the answer sets.
14:        for all $as \in AS$ do
15:            $abduciblesSet \leftarrow abduciblesSet \cup as$
16:            $alreadyComputed \leftarrow alreadyComputed \cup as$
17:        end for
18:    end for
19:    $abduciblesSet \leftarrow$ Least($abduciblesSet$)
20:    return $abduciblesSet.elements$
21: end function
```

case the program is satisfiable with answer sets of size 0, this means that there is no need to keep searching for abductive explanations since the empty explanation is sufficient, and we can stop the search. We now consider answer sets with 1 abducible, we remove the constraint for 0 abducibles and replace it with `:- c(X), X != 1`. There is one answer set, and thus an abductive explanation, {`e(b,c)`}, so a constraint of the form `:- e(b,c)` is added to the program. In this way, in the next iterations, we do not generate answer sets that contain this abducible, since it is already in a smaller explanation. That is, if a is an abductive explanation, all the subsequent explanations that contain a will be dominated, so they can be ignored. At the third iteration, we replace the constraint for 1 abducible with `:- c(X), X != 2`, while the constraints on the already discovered abducibles are kept. We obtain a new solution, {`e(d,e)`, `e(e,c)`}, so we add a new constraint to the program `:- e(d,e), e(e,c)`. We continue this process until considering 5 abducibles (all). At the end, we get {{`e(b,c)`}, {`e(d,e) e(e,c)`}} as abductive explanations. By default, abducibles not included into the abductive explanations are not selected. Note that, if we consider as minimality measure the number of abducibles in an answer set, we can solve this task by simply setting an optimization problem where the goal is to minimize the number of abducibles in the answer sets. That is, if we use, for example, the ASP system clingo [17], we can add the two rules:

```
c(C):- C = #count{X,Y : e(X,Y)}.
#minimize{C : c(C)}.
```

and get, as result, the minimal sets in terms of cardinality (`e(b,c)`).

In the next section we show how to extend this abductive framework in the case of *Probabilistic* Answer Set Programming.

4. Abductive Reasoning in Probabilistic Answer Set Programming

A *probabilistic* integrity constraint [8] is of the form

$$\Pi \leftarrow l_1, \ldots, l_n$$

where $\Pi \in]0,1]$ and each l_i is a literal (abducibles are allowed). Here, we also allow l_i to be an aggregate atom. Probabilistic facts and probabilistic integrity constraints identify the worlds, each of which may have multiple answer sets. We obtain a world by adding or not each ground probabilistic fact and each grounding of each probabilistic integrity constraint. For the grounding of the integrity constraints, we consider the standard concept of global and local variable [11]. In particular, a global variable of a rule is such that it appears in at least one literal not involved in aggregations. Variables only appearing in aggregates are called local. A ground instance of a rule with aggregates is obtained by first replacing global variables and then local variables with ground terms.

The probability of a world is given by the product of the atomic choices for the probabilistic facts with a factor Π_i for every probabilistic integrity constraint i selected and a factor $(1 - \Pi_j)$ for every probabilistic integrity constraint j not selected.

Definition 1 (Probabilistic abductive answer set program). *An answer set program P, a set of probabilistic facts F, a set of abducibles A, and a (possibly empty) set of probabilistic integrity constraints IC define a probabilistic abductive answer set program.*

Given a probabilistic abductive answer set program, the *lower joint* probability of the query q and the constraints IC given an explanation Δ is the sum of the probabilities of the worlds w where Δ is an explanation for the query q, all the constraints are satisfied, and q is true in all the answer sets. In formula:

$$\underline{P}(q, IC \mid \Delta) = \sum_{w:\forall m \in AS(P_w \cup \Delta), m \models q, m \not\models IC_{P_w}} P(w)$$

where P_w is the abductive answer set program identified by a word w and IC_{P_w} is the set of IC involved in P_w.

Finally, the goal of (cautious) abduction in probabilistic answer set programming is to find the minimal set of abducibles $\Delta \subseteq A$ such that $\underline{P}(q, IC \mid \Delta)$ is maximized, i.e., solve

$$\mathrm{least}(\arg\max_{\Delta} \underline{P}(q, IC \mid \Delta))$$

where, as in [8], the function least removes the sets that are not minimal. The main goal is to maximize the lower joint probability so, even if there are, for example, two solutions $\Delta' \subset \Delta''$ that yield respectively probabilities $P_{\Delta'} < P_{\Delta''}$, we only consider Δ'' as abductive explanation (despite being not minimal), since it gives the highest probability. Note that if a set Δ maximizes the lower joint probability, it may not maximize the upper joint probability. For example, if we consider the program:

```
abducible a.
abducible b.
0.5::fa.
0.5::fb.
query:- a,fa.
query;notquery:- b,fb.
```

the abductive explanation for the query `query` is $\Delta = \{$`a`$\}$ that yields a lower joint probability of 0.5, while the set of abducibles that maximizes the lower upper probability is $\{$`a,b`$\}$ that yields an upper probability of 0.75. The goal of (brave) abduction in probabilistic answer set programming can be defined in a similar way by considering the upper probability. However, here we focus on cautious abduction and every time we write abduction in probabilistic answer set programming we consider the goal of cautious abduction.

Example 2 (Probabilistic Smoke). *Suppose now that the relationships are uncertain. To model this, we add a probabilistic fact `fe/2` for every abducible in Example 1. Moreover, we suppose that the information provided by the integrity constraint is also uncertain, and has an associated probability of 0.2. The program became:*

```
abducible e(a,b). abducible e(b,c).
abducible e(a,d). abducible e(d,e).
abducible e(e,c).

0.5::fe(a,b). 0.5::fe(b,c). 0.5::fe(a,d).
0.5::fe(d,e). 0.5::fe(e,c).

friend(X,Y):- e(X,Y), fe(X,Y).
friend(X,Y):- e(Y,X), fe(Y,X).

smokes(b). smokes(d).

smokes(X) ; nosmokes(X):-
    friend(X,Y), smokes(Y).

0.2:- #count{X:nosmokes(X)} = N,
      #count{X:smokes(X)} = S,
      10*S < 8*(N+S).
```

The goal is to compute the abductive explanation for the query `smokes(c)`.

Probabilistic integrity constraints require a particular conversion. For every probabilistic IC of the form `p:- body` we add a probabilistic fact `p::f`, a rule `ic :- body`, and two constraints `:- f, ic`, and `:- not f, not ic` imposing respectively that, if the fact is selected, the constraint must be true and, if the fact is not selected, the constraint must be false. This new probabilistic fact is parsed as previously described.

To find the abductive explanations in PASP we modified the algorithm described in the previous section. We cannot impose the constraint that removes an already found explanation e since another explanation $e^{'} \supset e$ with a higher associated probability can exist. Moreover, we cannot add the constraint `:- not query`, since we need to consider the lower probability, and so ensure that the query is true in *all* the models for a world. Finally, we need to identify the choices made for the abducibles at each iteration and compute the probability for each world. If we consider Example 2 with the query `smokes(c)`, the only probabilistic abductive explanation is the set {`e(b,c)`,`e(d,e)`,`e(e,c)`} that gives a lower probability of 0.125. This process is summarized in Algorithm 2: first, the probabilistic facts, and the probabilistic integrity constraints are converted as previously explained (line 2). Then, we compute the minimal set of probabilistic and abducible facts (line 3) and add each fact of this minimal set into the program, as integrity constraint (line 5). The function AddConstraintSize adds a constraint to the program to limit the number of abducibles. The functions ExtractAbdChoices and ExtractWorlds respectively extracts the set of abducibles and the set of probabilistic facts for every answer set. The function ComputeContribution computes the contribution to both lower and upper probability [18] for every choice of abducibles and the function UpdateSet keeps track of the best solutions found so far. At the end of the loop, we check whether there are some solutions that are not minimal with the function Least and return the probabilistic abductive explanation and the probability range for the query.

Algorithm 2 Function AbductionPASP: computation of the probabilistic abductive explanations for a query $query$ and a PASP program $\mathcal{P}$.

```
1: function Abduction($query, \mathcal{P}$)
2:     $probFacts, abducibles, P_p \leftarrow$ ConvertProgram($\mathcal{P}$)
3:     $minSet \leftarrow$ ComputeMinimalSet($P_p \cup \{$:- $not\ query.\}$)
4:     for all $f \in minSet$ do
5:         $P_p \leftarrow P_p \cup \{$:- $not\ a.\}$
6:     end for
7:     $alreadyComputed \leftarrow \emptyset$
8:     $abduciblesSet \leftarrow \emptyset$
9:     for $i \in range(0, len(abducibles))$ do
10:        $P_p^c \leftarrow$ AddConstraintSize($P_p, i$)
11:        $P_p^{qc} \leftarrow P_p^c \cup \{q$:- $query., nq$:- $not\ query.\}$
12:        $projectSet \leftarrow probFacts \cup abducibles \cup \{q\} \cup \{nq\}$
13:        $AS \leftarrow$ ProjectSolutions($P_p^{qc}, projectSet$)
14:        for all $as \in AS$ do
15:            $wa \leftarrow$ ExtractAbdChoices($as$)        ▷ Identify choices for the abducibles
16:            $wp \leftarrow$ ExtractWorlds($wa$)        ▷ Extract worlds
17:            $contributionsList \leftarrow$ ComputeContribution($wp$)
18:            $abduciblesSet \leftarrow$ UpdateSet($contributionsList$)
19:        end for
20:    end for
21:    $abduciblesSet \leftarrow$ Least($abduciblesSet$)
22:    return $abduciblesSet.lp$, $abduciblesSet.up$, $abduciblesSet.elements$
23: end function
```

5. Experiments

To evaluate our approach, we ran the proposed algorithm on a computer with Intel® Xeon® E5-2630v3 running at 2.40 GHz with 16Gb of ram and a time limit of 8 hours. We used the ASP solver clingo [17]. We tested three datasets, both for ASP and PASP[1].

The first dataset, `clauses`, encodes a domain with an increasing number of abducibles `ab/1`, a clause for every one of them and a constraint imposing that at least two abducibles should be selected. The structure for the program of size 4 (where the size indicates the number of abducibles) for ASP is the following:

```
abducible ab(1). abducible ab(2).
abducible ab(3). abducible ab(4).
qry:- ab(1). qry:- ab(2). qry:- ab(3). qry:- ab(4).
:- #count{X : a(X)} = C, C != 2.
```

The query is `qry`. For the PASP version we considered deterministic and probabilistic integrity constraints. In the first case half of the `ab/1` atoms are abducibles and half are probabilistic facts with an associated probability of 0.5. In the second case, the IC has an associated probability of 0.5. In both cases, the integrity constraints only involves the number of abducibles that can be selected. The remaining part of the program is the same. The goal of this experiment is to test the inference time of the algorithm on programs with an increasing number of clauses that must be considered for the computation of the (probabilistic) abductive explanation.

Results are shown in Figure 2. The execution times for PASP with deterministic and probabilistic integrity constraints are similar, and, in both cases, we get a memory error for size 26. Instead, the time required for the computation of abductive explanations in ASP is negligible with respect to PASP.

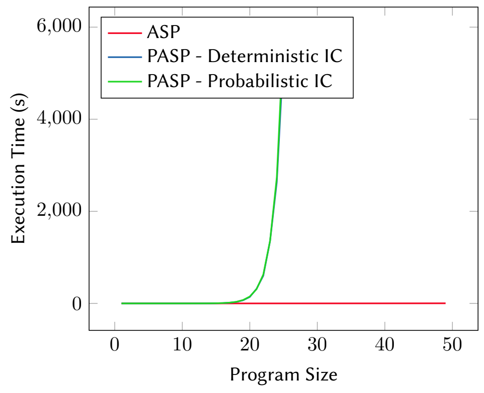


Figure 2: Inference times for the `clauses` experiments.

The second dataset, `bird`, encodes a small biological domain composed of birds. Each bird can either fly or not fly, and there are at least 60% of birds that fly. An example of program of size 4 (with 4 birds) is the following:

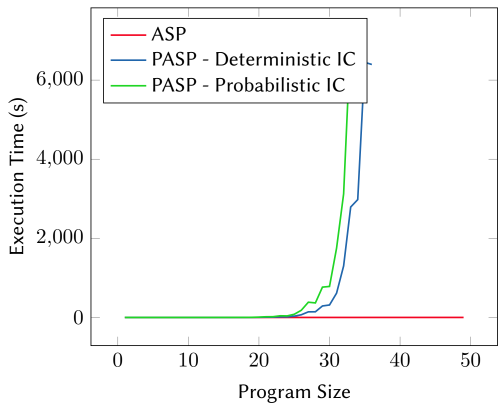
[1]Source code and datasets available at: https://github.com/damianoazzolini/pasta.

```
bird(1). bird(2). bird(3). bird(4).
fly(X);nofly(X):- bird(X).
:- #count{X:fly(X),bird(X)} = FB,
   #count{X:bird(X)} = B, 10*FB<6*B.
```

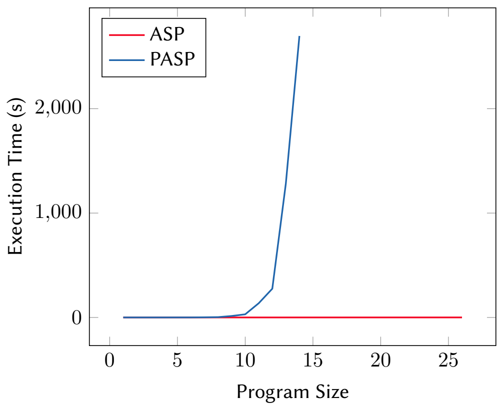
the query is `fly(1)`.

For the ASP version, we considered an increasing number of `bird/1` facts as abducibles. For PASP with deterministic constraints 1/3 of the `bird/1` facts are certain, 1/3 are probabilistic with an associated probability of 0.5, and 1/3 are abducibles. For PASP with probabilistic integrity constraints the split is the same, but the constraint imposing that 60% of birds fly has an associated probability of 0.5.

Results are shown in Figure 3a. As expected, the introduction of a probabilistic integrity constraint in the PASP program increases the execution time with respect to the same program with a deterministic IC, since an additional probabilistic fact must be considered. As before, the execution time to compute the abductive explanation in ASP is constant and almost negligible up to size 50.



(a) Inference times for the `bird` experiments.



(b) Inference times for the `smoke` experiments.

Figure 3: Results for the `bird` and `smoke` experiments.

A third dataset, `smoke`, encodes a network where nodes represent people and edges represent relationships. A person can either smoke or not smoke. Other people can either smoke or not smoke if they are influenced by others. This benchmark has no integrity constraints. An example program of size 4 is the following:

```
smokes(X) ; nosmokes(X) :- smokesFact(X).
smokes(X) ; nosmokes(X) :- smokes(Y), influences(X,Y).

smokesFact(1). smokesFact(2). smokesFact(3).
smokesFact(4). influences(0,1). influences(0,2).
influences(0,3). influences(1,3).
```

The goal is to compute the (probabilistic) abductive explanations for the query `smokes(1)`.

For abduction in ASP, half of the `smokesFact/1` facts are abducibles and half are certain. Similarly, half of the `influences/2` facts are abducibles and half are certain. For abduction in PASP, half of the `smokesFact/1` facts are probabilistic with an associated probability of 0.5 and half are certain, and half of the `influences/2` facts are abducibles and half are certain. The generation of `influences/2` facts follow a Barabási-Albert model (we used the method `barabasi_albert_graph` from the networkx [19] python package).

Results are shown in Figure 3b. The inference time trend in the two cases coincides with the previous experiments.

The gap of execution time between abduction in ASP and PASP is too big to be analyzed. Thus, we decided to run a separate experiment for abduction in ASP, with larger programs for all the datasets. Results are shown in Figure 4. The `bird` and `smoke` datasets have similar running time while the `clauses` one is the slowest among the three: this is probably because each program has a significant number of abductive explanations: for example, the program of size 10 has 45 while the program of size 360 has 64620 abductive explanations.

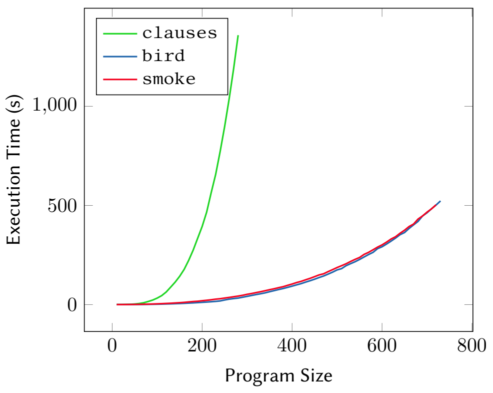


Figure 4: Inference times for abduction in ASP with programs of different sizes for the 3 datasets.

6. Related Work

Abduction and Answer Set Programming are strongly related [20]. In [21] the authors propose a specialized framework to perform abductive reasoning under the stable model semantics by defining a special T_p operator. Differently from them, we leverage an existing ASP solver (clingo) to perform abduction, and do not develop specialized operators. ABDUAL [22], later refined in TABDUAL [23], is a framework that performs abduction in the context of well-funded semantics with the capability to compute stable models. It is implemented in XSB Prolog [24] and leverages common Logic Programming techniques such as tabling. Differently from them, we use an ASP framework, easily supporting the whole ASP syntax. Recently, the authors of [8] proposed an algorithm to perform abduction in probabilistic logic programs under the Distribution Semantics. To the best of our knowledge, no existing frameworks can perform abduction in Probabilistic Answer Set Programming under the Credal Semantics.

7. Conclusions

In this paper we proposed a new algorithm to perform abduction in Answer Set Programming and Probabilistic Answer Set Programming under the Credal Semantics. For the former, the goal is to find the minimal set, where minimality is defined in terms of set inclusion, of abducible facts that explains a query, i.e., such that the query has at least one answer set. For the latter, the goal is to find the minimal set which maximizes the lower joint probability of the query and the constraints. Results on three datasets show that abduction for ASP is orders of magnitude faster than for PASP, since the generation of all the worlds is not needed. As future work, we plan to apply approximate inference [25] to speed up the computation in PASP. A further direction for future work could consist in better exploring the relation between lower and upper probability for abduction in PASP.

References

[1] A. C. Kakas, P. Mancarella, Abductive logic programming, in: V. W. Marek, A. Nerode, D. Pedreschi, V. S. Subrahmanian (Eds.), Proceedings of the Workshop Logic Programming and Non-Monotonic Logic, Austin, TX, USA, November 1–2, 1990, 1990, pp. 49–61.

[2] A. C. Kakas, P. Mancarella, Database updates through abduction, in: Proceedings of the 16th VLDB, Morgan Kaufmann, 1990, pp. 650–661.

[3] J. W. Lloyd, Foundations of Logic Programming, 2nd Edition, Springer, 1987.

[4] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[5] L. De Raedt, P. Frasconi, K. Kersting, S. Muggleton (Eds.), Probabilistic Inductive Logic Programming, volume 4911 of *LNCS*, Springer, 2008.

[6] F. Riguzzi, Foundations of Probabilistic Logic Programming: Languages, semantics, inference and learning, River Publishers, Gistrup, Denmark, 2018.

[7] T. Sato, A statistical learning method for logic programs with distribution semantics, in: L. Sterling (Ed.), ICLP 1995, MIT Press, 1995, pp. 715–729. doi:10.7551/mitpress/4298.003.0069.

[8] D. Azzolini, E. Bellodi, S. Ferilli, F. Riguzzi, R. Zese, Abduction with probabilistic logic programming under the distribution semantics, International Journal of Approximate Reasoning 142 (2022) 41–63. doi:10.1016/j.ijar.2021.11.003.

[9] F. G. Cozman, D. D. Mauá, On the semantics and complexity of probabilistic logic programs, J. Artif. Intell. Res. 60 (2017) 221–262. doi:10.1613/jair.5482.

[10] F. G. Cozman, D. D. Mauá, The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference, Int. J. Approx. Reason. 125 (2020) 218–239. doi:10.1016/j.ijar.2020.07.004.

[11] M. Alviano, W. Faber, Aggregates in answer set programming, KI-Künstliche Intelligenz 32 (2018) 119–124. doi:10.1007/s13218-018-0545-9.

[12] W. Faber, N. Leone, G. Pfeifer, Recursive aggregates in disjunctive logic programs: Semantics and complexity, in: European Workshop on Logics in Artificial Intelligence, Springer, 2004, pp. 200–212. doi:10.1007/978-3-540-30227-8_19.

[13] L. De Raedt, A. Kimmig, H. Toivonen, Problog: A probabilistic prolog and its application in link discovery, in: M. M. Veloso (Ed.), IJCAI, 2007, pp. 2462–2467.

[14] A. Van Gelder, K. A. Ross, J. S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (1991) 620–650.

[15] T. Eiter, G. Gottlob, The complexity of logic-based abduction, J. ACM 42 (1995) 3–42. doi:10.1145/200836.200838.

[16] M. Gebser, B. Kaufmann, T. Schaub, Solution enumeration for projected boolean search problems, in: W.-J. van Hoeve, J. Hooker (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer-Verlag, 2009, pp. 71–86. doi:10.1007/978-3-642-01929-6_7.

[17] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot asp solving with clingo, Theory and Practice of Logic Programming 19 (2019) 27–82. doi:10.1017/S1471068418000054.

[18] D. Azzolini, E. Bellodi, F. Riguzzi, Statistical statements in probabilistic logic programming, in: G. G. D. Inclezan, M. Maratea (Eds.), 16th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022), 2022.

[19] A. A. Hagberg, D. A. Schult, P. J. Swart, Exploring network structure, dynamics, and function using networkx, in: G. Varoquaux, T. Vaught, J. Millman (Eds.), Proceedings of the 7th Python in Science Conference, Pasadena, CA USA, 2008, pp. 11–15.

[20] M. Denecker, A. Kakas, Abduction in logic programming, Lecture Notes in Computer Science 2407 (2002) 402–436.

[21] K. Inoue, C. Sakama, Computing extended abduction through transaction programs, Ann. Math. Artif. Intell. 25 (1999) 339–367. doi:10.1023/A:1018926021566.

[22] J. J. Alferes, L. M. Pereira, T. Swift, Abduction in well-founded semantics and generalized stable models, CoRR cs.LO/0312057 (2003).

[23] A. Saptawijaya, L. M. Pereira, Tabdual: a tabled abduction system for logic programs, FLAP 2 (2015) 69–124.

[24] T. Swift, D. S. Warren, XSB: Extending prolog with tabled logic programming, Theor. Pract. Log. Prog. 12 (2012) 157–187. doi:10.1017/S1471068411000500.

[25] D. Azzolini, F. Riguzzi, E. Lamma, F. Masotti, A comparison of MCMC sampling for probabilistic logic programming, in: M. Alviano, G. Greco, F. Scarcello (Eds.), Proceedings of the 18th Conference of the Italian Association for Artificial Intelligence (AI*IA2019), Rende, Italy 19-22 November 2019, volume 11946 of *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, 2019, pp. 18–29. doi:10.1007/978-3-030-35166-3_2.

Logic Programming library for Machine Learning: API design and prototype

Giovanni Ciatto[1], Matteo Castigliò[3] and Roberta Calegari[2]

[1]Departement of Computer Science and Engineering (DISI), Alma Mater Studiorum—Università di Bologna
[2]Alma Mater Research Institute for Human Centered AI (AlmaAI), Alma Mater Studiorum—Università di Bologna
[3]Master Student at DISI

Abstract
In this paper we address the problem of hybridising symbolic and sub-symbolic approaches in artificial intelligence, following the purpose of creating flexible and data-driven systems, which are simultaneously comprehensible and capable of automated learning. In particular, we propose a logic API for supervised machine learning, enabling logic programmers to exploit neural networks – among the others – in their programs. Accordingly, we discuss the design and architecture of a library reifying APIs for the Prolog language in the 2P-Kt logic ecosystem. Finally, we discuss a number of snippets aimed at exemplifying the major benefits of our approach when designing hybrid systems.

Keywords
Logic programming, Machine Learning, API, 2P-Kt

1. Introduction

Symbolic and sub-symbolic artificial intelligence (AI) are complementary under several perspectives [1, 2]. For this reason, many recent contributions from the literature are discussing the possible frameworks for their integration and hybridisation [3, 4, 5, 6]. However, what is currently slowing down scientific progress in this context is not the lack of ideas concerning *how* such integration and hybridisation may occur. Conversely, the bottleneck is caused by the lack of suitable technologies enabling and easing the experimentation of integrated or hybrid systems. Logic-based technologies are in fact technological islands, for which poor care is given to the construction of bridges with the rest of the AI land.

Accordingly, in this paper, we address the issue of supporting machine learning (ML) – and, in particular, neural-networks (NN) based training and inference – in logic programming (LP). We do so by designing and prototyping a logic based API for machine learning. Along this line, our contribution is twofold: *(i)* we let logic programmers exploit the benefits of sub-symbolic AI, and, in particular, neural networks; and *(ii)* we enable the practical experimentation of hybrid

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
*Corresponding author.
giovanni.citatto@unibo.it (G. Ciatto); matteo.castiglio@studio.unibo.it (M. Castigliò); roberta.calegari@unibo.it (R. Calegari)
https://about.me/gciatto (G. Ciatto); http://robertacalegari.apice.unibo.it (R. Calegari)
0000-0002-1841-8996 (G. Ciatto); 0000-0003-3794-2942 (R. Calegari)

systems—involving both logic and neural processing of data.

Our logic-based API for ML consists of a set of logic predicates enabling the representation, training, testing, and exploitation of sub-symbolic predictors in LP—possibly, out of data expressed in logic form. In other words, our API lets logic programmers use neural networks in their programs – e.g. to train or exploit classifiers or regressors – without requiring them to abandon the logic realm. Of course, to make this possible, our API supports the whole gamma of low level tasks that are commonly involved in an ML workflow—including, but not limited to, data preprocessing, cross-validation, etc.

Technically, we prototype our API via a logic library – namely, the ML-Lib – targetting the 2P-Kt ecosystem [7], the JVM platform, and the Prolog language [8]. DeepLearning4J [9] is the underlying library we leverage on in this paper. However, our design is general enough to support other libraries and, possibly, different platforms—e.g. Tensorflow [10] over Python.

Arguably, our work represents the first step towards a wider degree of interoperability among symbolic- and sub-symbolic AI. In fact, one the long run, we aim to enable the design and construction of hybrid systems, fruitfully and dynamically combining the major advantages of both approaches to artificial intelligence by mixing inferential (via LP) and intuitive (via NN) reasoning capabilities. Along this path, the proposed API is a key enabling factor, as it supports the creation of logic-based inferential engines which are capable of learning from data via state-of-the-art mechanisms. Dually, by supporting the training of neural networks from logic data, our API can also be considered a tool for endowing sub-symbolic predictors with prior, high-level knowledge.

2. Logic library for ML: goals

To properly design a logic library for dealing with hybrid reasoning, some basic goals to achieve should be taken into account: namely, *(i)* enable hybrid reasoning, *(ii)* full support of declarative ML, *(iii)* enable the exploitation of symbolic data sources (in addition to the others), *(iv)* make it possible to select a model via resolution. It is worth mentioning that, each one of these goals comes along with some of the benefits of hybridization, discussed in detail in [2]. In the following details about these goals are discussed.

Hybrid reasoning. Automatic reasoning may greatly benefit from sub-symbolic AI to overcome its inherent crispness. Fuzzy data could then be suitably and coherently processed by a sub-symbolic predictor as part of a wider symbolic resolution process. To make this possible, sub-symbolic predictors should be representable, trainable, and queryable as any other logic predicate, without requiring the semantics of logic resolution to be affected. Consequently, logic programs should be endowed with ad-hoc predicates and syntactical categories, aimed at representing and manipulating sub-symbolic predictors and data.

Declarative ML. Declarative ML is a paradigm by which data scientists' code should only specify *what* an ML workflow should do, by leaving the underlying platform in charge to understand *how*. This is partially supported by the current practice of data science which relies on high-level languages (e.g. Python) and libraries of elementary components to be composed (e.g. Scikit-Learn [11]). However, the solutions proposed so far do not leverage inherently declarative frameworks like LP, but rather object-oriented languages—requiring imperative statements. Hence, to support the declarative expression of an ML workflow in the LP framework, a new logic API is required.

Symbolic data sources. Logic knowledge bases are a peculiar way of collecting knowledge. Unlike datasets and DBMS, they represent information in symbolic form, via – possibly *intensional* – logic formulæ. Hence, they can virtually represent any sort of datum – be it atomic, compound or structured – via a concise (yet very expressive) language, while possibly saving space. Accordingly, when combining LP with ML, knowledge bases should be exploitable as data sources as well—other than ordinary CSV files or relational databases.

Model selection via resolution. Logic resolution essentially consists of a search procedure aimed at finding solutions in a proof tree. This could be applied to a common step of any ML workflow—namely *model selection*. There, data scientists must assess several predictor families, to select the one which is better suited for the learning task at hand. Then, they must search for the best hyper-parameters for the selected family of predictors. All such choices involve several sorts of predictors, with possibly different hyper-parameters, to be trained and compared—either in an orderly fashion or in parallel. LP naturally captures the non-deterministic exploration of a space of possible choices. Hence it is well suited to both declaratively represent and implement model selection.

3. ML: key aspects to be supported

To support the aforementioned goals, logic APIs must cover the full gamma of aspects involved in any possible ML workflow, detailed and discussed in this section.

Briefly speaking, an ML workflow is the process of producing a suitable predictor for the available data and the learning task at hand, following the purpose of later exploiting that predictor to draw analyses or to drive decisions. Each ML workflow can be conceived as composed of six major phases – elicited in section 3.1 –, each one involving a number of activities—elicited in section 3.2. Enumerating and defining all possible phases and activities is of paramount importance, as any API for ML should support them all.

3.1. ML phases

From a coarse-grained perspective, a machine learning workflow is composed of six major phases, detailed in the following.

Dataset loading. The first step of any ML workflow consists of loading that dataset in memory for later processing. To support such a step, ML frameworks come with ad-hoc functionalities aimed at loading the dataset by reading a file from the local file system, fetching it from the Web, or querying a DBMS. These usually come in the form of either classes or functions, coherently w.r.t. the object-oriented nature of mainstream ML frameworks. Accordingly, the logic API for ML should expose ad-hoc *predicates* to serve the same purposes. Furthermore, however, it should also support the loading of datasets out of logic theories of facts and rules.

Data pre-processing. Raw datasets are often inadequate to favour predictors' training. Hence, dataset pre-processing is commonly practised to increase the effectiveness of any subsequent training phase. Most common bulk operations of pre-processing are: *(i)* homogenize the variation ranges of the many features sampled by the dataset, *(ii)* detect irrelevant features and remove them, *(iii)* construct relevant features by combining the existing ones, *(iv)* encoding non-numeric features into numeric form, and *(v)* horizontal (by row) or vertical (by column) partitioning of the dataset. In particular, the purpose *(v)* is of paramount importance, as it supports the *test set separation* as well as splitting input-related columns from output-related ones—fundamental operation to enable validation and testing and to support training respectively.

Predictor selection and definition. Many sorts of predictors could be used in principle to perform supervised learning—e.g. neural networks, decision trees, support vector machines, etc. A preliminary phase to select the best predictor is a common phase in virtually any ML workflow. Once a particular sort of predictor has been chosen, a way to specify the shape the to-be-trained predictor should have is required. Of course, such specification should take into account the schema of the input data, as well as the schema of the expected outcomes to be produced by the predictor. Finally, *hyper-parameters* of the selected algorithm need to be tuned.

Accordingly, the logic API for ML should support the specification of as many sorts of predictors as possible, as well as their parametrisation. Once again, predicates should be defined to serve this purpose. In particular, at least one ad-hoc predicate should be defined for each sort of predictor to be supported, carrying as many arguments as the possible hyper-parameters that could be specified for that sort of predictor. In case hyper-parameters cannot be conveniently represented as raw logic types (numbers or strings), ad-hoc predicates should be provided as well for constructing structured hyper-parameters values.

Training. Predictors' training plays a pivotal role in ML workflows. This is the phase where predictors are fit on the available data or, in other words, automated learning actually occurs. Generally speaking, training can be modelled in LP as a single predicate, mapping untrained predictors into trained ones, possibly via a number of learning parameters (e.g. learning rate or momentum for NN, or maximum depth for DT), or stopping criteria (e.g. max epochs for NN, or max depth for DT), other than, of course, the data to be used for training. Once again, several ad-hoc predicates should be defined to support structured parameters or stopping criteria in the logic API for ML. Furthermore, regardless of its shape, the training predicate should accept some arguments aimed at specifying whether the columns of the training set should be considered as inputs or outputs.

Inference. Inference is commonly the last phase of any ML workflow. Here, trained predictors are used to draw predictions on new data—i.e. different data w.r.t. the training set. In most common cases, predictions attempt to solve classification or regression problems. In any case, yet another general predicate should be added to our logic API for ML to support drawing predictions out of a trained predictor and a set of raw data (or a single datum). Ad-hoc predicates may be provided as well to explicitly model higher-level tasks, such as classification and regression. Finally, it should be possible to store, retrieve, and re-apply any pre-processing procedure possibly defined before training, to the raw data for which predictions should be drawn—in order to make it acceptable for the predictor as an input.

Validation. Validation is the *penultimate* step of any ML workflow: it follows the training and precedes the exploitation. It is here discussed as last because it technically relies on the capability of drawing predictions via trained predictors—which is treated in the paragraph above.

Generally speaking, validation attempts to measure the predictive performance of a trained predictor, with the purpose of assessing if and to what extent it will generalise to new, unseen data. To this end, the predictor is tested against the test set—that is, a collection of unseen data, for each expected predictions exist. The discrepancy (or similarity) among the actual and expected predictions is then measured via ad-hoc scoring functions (a.k.a. measures), resulting in a performance assessment for the trained predictor. Many measures may be used to assess classifiers (e.g. accuracy, F1-score, etc.) and as many to assess regressors (e.g. MAE, MSE, R^2, etc.). Hence, to support validation, the logic API for ML should provide predicates to compute each possible measure.

3.2. ML activities, per phase

Here we elicit the many activities involved in each phase of any ML workflow, and we describe them from a computational perspective—i.e. in terms of the sorts of entities (a.k.a. data types) they accept as input or produce as output (manipulate, for short).

Entities. We start our discussion by identifying the five major sorts of entities each activity may manipulate.

- *Value:* a scalar, vectorial, matrix, or tensorial datum from a given domain (e.g. integer or real numbers, or vectors of integer or real numbers, as well as a string, a table, a time series, etc.).
- *Schema:* a concise and formal description of a domain (i.e. a set of values). For scalar values, schemas are essentially data types (e.g. integers, reals, strings, etc.), while for non-scalar data they carry information about the name, index, and type of each single scalar component.
- *Dataset:* a collection of values matching a particular schema (supposed to be known).
- *Transformation:* any operation aimed at transforming an entity dataset into another other—commonly, a dataset into either another dataset (e.g. normalization, standardization, etc.) or a value (e.g. max, min, average, etc.) From an algebraic perspective, it is a function. From a computational perspective, it is an algorithm.

- *Predictor:* a stateful computational entity capable of *(i)* drawing predictions (i.e. outputting values) out of (possibly unseen) input values, according to its internal state *(ii)* updating its internal state according to a dataset (to improve future predictions)

Any logic-based API for ML should support the representation, combination, and manipulation of entities of these kinds.

Activities. Each phase of the ML workflow is then characterised by a specific set of activities possibly manipulating entities of any of these sorts. A logic-based API for ML should support them as well. Accordingly, in the remainder of this section, we describe such activities along with the entities they operate upon. In doing so, we partition activities w.r.t. the ML phase they are most commonly exploited into.

Dataset loading. The main activities to support the loading of a dataset into a solver's memory, and its preparation for subsequent processing are

- *Dataset loading:* operation of loading a dataset from either a value – representing either a local or remote file –, or from a Prolog theory
- *Schema declaration:* operation of constructing a representation for a given schema
- *Target features declaration:* operation of tagging a portion of the features of some schema as either inputs or outputs (a.k.a. targets)
- *Dataset splitting:* operation of horizontally partitioning a dataset into two or more smaller datasets.

Dataset pre-processing. Here, they may be willing to define transformations or cascades of transformations (pipelines, henceforth) to be eventually applied to datasets:

- *Transformation declaration:* operation of declaratively encoding a transformation operation to be applied to all data in a dataset
- *Pipeline composition:* operation of declaratively constructing a composite transformation as a cascade of simpler transformations
- *Transformation application:* operation of actually constructing a new dataset from a prior dataset and a transformation

Predictor selection and definition. The next phase involves the *definition* of one or more predictors via a unique meta-activity, namely:

- *Predictor declaration:* operation of constructing a representation for a particular predictor, which implies choosing the predictor family and specifying actual values for its hyper parameters

Training. Eventually, declared predictors may enter the *training* phase, meaning that their learning from data should be triggered. This can be achieved via yet another activity, namely:

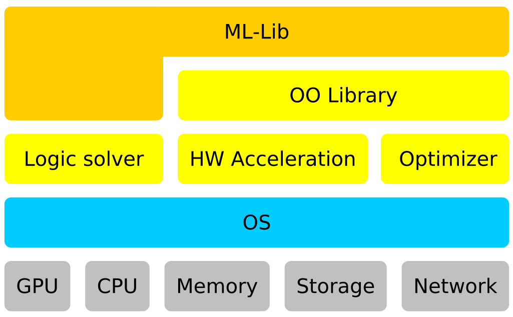


Figure 1: Layered view of the proposed ML-Lib. An OO library is assumed behind the scenes, providing high-level abstraction to optimize ML predictors, possibly via HW acceleration.

- *Predictor fitting w.r.t. a training set of data:* operation of fitting a predictors' internal parameter on some provided training data

Inference. Once in their *inference* phase, trained predictors may eventually be exploited to draw predictions. The corresponding activity is:

- *Predictor querying:* operation where (possibly unseen) values are provided to some trained predictor as a query, and the resulting values are interpreted as predictions

Validation. Finally, in the *validation* phase, trained predictors should be assessed by measuring their performance w.r.t. some test data This is yet another meta-activity, with several possible variants depending on the particular measure being exploited:

- *Predictor scoring:* operation of computing a scoring value out of a trained predictor, a test dataset, and a scoring function

4. ML-Lib Overview and Architecture

This section discusses the design of ML-Lib, the logic programming library reifying the logic API for ML reifying the meta-model discussed above. The overall architecture is depicted in fig. 1. The ML-Lib assumes a goal-oriented logic solver being in place, where ordinary logic programs can be executed. Thanks to the ML-Lib, these logic programs may also exploit a number of predicates for training and using ML predictors—other than any other entity involved in the process.

In the backend, the library assumes an underlying object-oriented (OO) library providing high-level ML abstractions, such as datasets, predictors, and so on. Examples of these libraries may be for instance Keras [12] or DeepLearning4J [9]. The OO library may in turn be backed by an optimizer taking care of making training and data management effective on the available hardware—and possibly exploiting hardware acceleration. In practice, software such as Theano, Caffe, or Tensorflow may serve this purpose. Actual technological choices may finally depend on the particular runtime platforms being targeted. For instance, targeting the JVM may

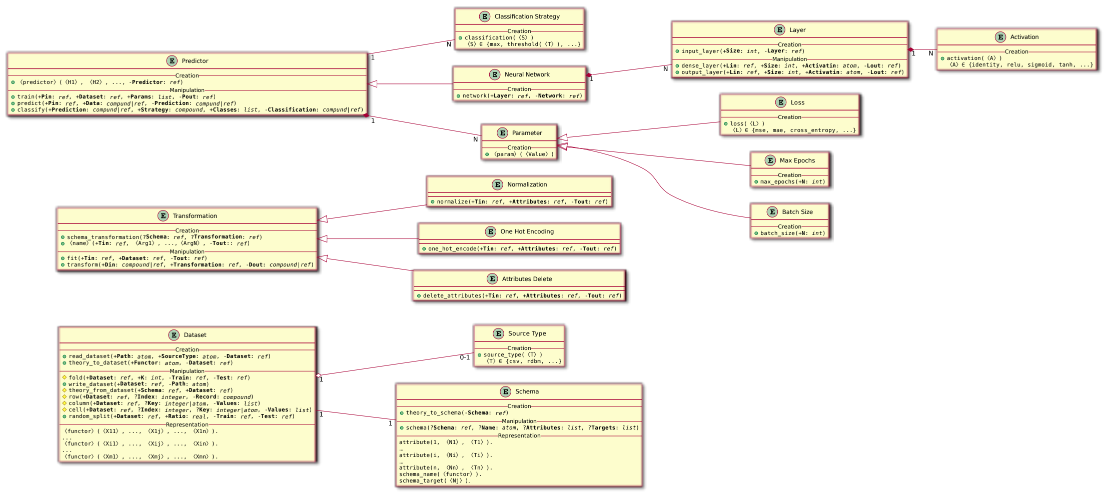


Figure 2: Overview of our ML-Lib design. The chart represents the many entities logic programmers may exploit via our ML-Lib, and the many predicates supporting their creation, manipulation, or representation. Predicates are depicted with either a yellow diamond in case they are non-deterministic (a.k.a. backtrackable), or a green circle otherwise.

imply DeepLearning4J to be exploited, while targetting Python may exploit both Keras and Tensorflow. However, while technological choices are contingent and subject to change, the overall architecture is meant to support the implementation of the ML-Lib as a façade towards the underlying OO library, regardless of what it is.

At the functional level, the design of the ML-Lib is provided in terms of logic predicates acting on the above defined entities. Details about the predicates are provided in the supplementary material. Figure 2 provides an overview of these predicates, grouped by entities.

5. ML-Lib Examples

Here we discuss the usage of the ML-Lib to serve the purposes described in section 2.

From an LP perspective, our examples assume the existence of a logic solver/language exploiting some implementation of the ML-Lib. For the sake of simplicity, we assume a Prolog solver is employed. Examples consist of Prolog scripts, possibly involving standard Prolog predicates.

From an ML perspective, our examples assume a very simple scenario where a neural-network classifier is trained on the well known Iris dataset [13]. The resulting NN is then exploited to write a simple hybrid predicate aimed at classifying unseen Iris instances.

Declarative ML. Declarativeness is a key benefit of our symbolic approach to ML. The ML-Lib supports declarative ML in several ways, as exemplified by listings 1, 2, 3, and 5.

In particular, listing 1 shows how the schema and data entries of the Iris dataset can be treated in logic. Notably, the Iris data set contains 150 rows describing as many individuals

```prolog
% schema declaration
attribute(0, sepal_length, real).
attribute(1, sepal_width, real).
attribute(2, petal_length, real).
attribute(3, petal_width, real).
attribute(4, species, categorical([setosa, versicolor, virginica])).
schema_target([species]).
schema_name(iris).

% reading schema from theory
iris_schema(S) :- theory_to_schema(S).

% dataset loading
iris_dataset(D) :-
    read_dataset('/path/to/iris.csv', csv, D).
```

Listing 1: Dataset loading from file

```prolog
%  declaring & fitting the preprocessing pipeline
preprocessing_pipeline(Dataset, Schema, Pipeline) :-
    schema_transformation(Schema, Step0),
    normalize(Step0, [petal_width, petal_length, sepal_width, sepal_length], Step1),
    one_hot_encode(Step1, [species], Step2),
    fit(Step2, Dataset, Pipeline).
```

Listing 2: Pre-processing pipeline

```prolog
% neural network declaration
multi_layer_perceptron(Nin, Nhidden, Nout, NN):-
    input_layer(Nin, IL),
    hidden_layer(IL, Nhidden, H),
    output_layer(H, Nout, softmax, O),
    neural_network(O, NN).

hidden_layer(L, [], L).
hidden_layer(L, [N | M], H) :-
    dense_layer(L, N, relu, L1), hidden_layer(L1, M, H).
```

Listing 3: Neural network structure declaration

of the Iris flower. For each exemplary, 4 continuous input attributes – *petal* and *sepal width* and *length* – are recorded, other than a categorical target attribute—denoting the actual Iris *species*. There are three particular species of Iris in this data set – namely, Setosa, Virginica, and Versicolor –, and the 150 examples are evenly distributed among them—i.e., there are 50 instances for each class. The Prolog script describes the Iris dataset's schema in clausal form, as discussed in appendix A.1.1. It also declares two predicates – namely, `iris_schema/1` and `iris_dataset/1` – aimed at letting the logic programmer retrieve either the schema or its dataset in object form. More precisely, `iris_schema/1` attempts to read the schema from the local theory, while `iris_dataset/1` attempts the load the dataset from a CSV file. Listing 4 (presented later in this section) reports a similar scenario where the dataset as well is loaded from the local theory.

Listing 2 exemplifies the declaration of a pre-processing pipeline aimed at normalising the input attributes of any `Dataset` having the same `Schema` of Iris, other than one-hot encoding its output attributes. The resulting `Pipeline` is then fitted against the provided `Dataset`, and bound to the corresponding output argument.

In turn, listing 3 presents a general purpose predicate aimed at defining multi-layered perceptron predictors with an arbitrary amount of hidden layers. This is enabled by the `multi_layer_perceptron/4` predicate, which requires the caller to provide the number of neurons to be instantiated for *(i)* the input layer (`Nin`), *(ii)* the output layer (`Nout`), and *(iii)* for each hidden layer (`Nhidden`). Notably, `Nhidden` should consist of a list in integers, denoting the number of neurons for each hidden layer – from the outermost to the innermost –, while the total amount of integers corresponds to the number of hidden layers. The resulting neural network predictor is then bound to the `NN` output argument. So, for instance, a NN having 4 input neurons, 2 hidden layers with 5 and 7 neurons respectively, and 3 output neurons can be declared as follows:

```
multi_layer_perceptron(4,[5, 7],3,NN)
```

Finally, listing 5 declares an end-to-end ML workflow aimed at selecting and training the best NN architecture to tackle Iris classification. It is worth noting that the declarative nature of the script can be regarded as a formal – yet human-readable – specification of a classifier training workflow.

Symbolic data sources. As highlighted above it may be useful to perform ML upon data expressed in logic form. This requires logic theories to act as symbolic data sources. ML-Lib supports such scenario, as exemplified in listing 4. The script is assumed to replace listing 4 in those situations where the Iris dataset is logically described in the clausal form. Here, the `iris_dataset/1` attempts to load the data from the local theory instead of a file.

Model selection via resolution. The automatic exploration of a search space subtended by logic resolution could be exploited to perform model selection. Indeed, model selection essentially consists of an exploration of the hyper and learning parameters space, looking for the best possible values—i.e. those hyper and learning parameters assignments corresponding to well-performing predictors on the available training set.

Accordingly, the ML-Lib supports expressing and performing model selection in logic (listing 5). There hyper, learning, and workflow parameters are expressed as logic facts, and the `params/2` predicate is defined to enumerate all possible combinations of theirs—e.g. via Prolog's backtracking mechanism. The `model_selection/5` predicate is in charge of stepping through all such parameters with the purpose of selecting, and training all corresponding NN predictors which attain a sufficiently high predictive performance—denoted by the `target_performance/1` fact. For each trained predictor, the predicate outputs not only a

```
/* attributes definition here */

% dataset definition
iris(5.1, 3.2, 1.4, 0.2, setosa).
iris(4.9, 3, 1.7, 1.2, versicolor).
iris(5.9, 3.4, 1.1, 0.9, virginica).
/*... other entries here...*/

% reading schema from theory
iris_schema(S) :- theory_to_schema(S).

% reading dataset from theory
iris_dataset(D) :- iris_schema(S), theory_to_dataset(S, D).
```

Listing 4: Dataset loading from the local theory

reference to the `Predictor` itself, but also its `Performance`, and the affine `Transformation` to be applied to each datum for which predictions should be drawn using that predictor. The predicate `model_selection/5` works by

1. splitting the provided `Dataset` into a `TrainingSet` and a `TestSet`, according to a split ratio (`R`) declared by the `test_percentage/1` fact
2. declaring and fitting a pre-processing `Transformation` aimed at normalising the `TrainingSet`'s input attributes, and one-hot encoding its output attributes
3. applying such `Transformation` to the `TrainingSet`, hence producing a `ProcessedTrainingSet`
4. stepping through all possible hyper (`HyperParams`) and learning (`LearnParams`) parameters combinations,
5. training each corresponding predictor, via 10-fold cross validation (CV), and computing its average validation-test performance (`P`)
6. skipping each hyper and learning parameters combination such that the average performance `P` is lower than the target performance `T`
7. re-training a full-fledged MLP on the whole `TrainingSet`, for each parameters combination such that `P >= T`
8. testing that MLP against the `ProcessedTestSet` – obtained by applying `Transformation` to the `TestSet` –, thus computing the MLP actual `Performance`

In other words, the `model_selection/5` represents a declarative, and pretty general, workflow for model selection—which may be adapted to other supervised learning tasks with minimal changes. Further details about the many predicates exploited in this example are provided in the supplementary material.

```
 1  /* Hyper paramenters */
 2  hidden_layers([10]). hidden_layers([20, 10]).
 3  hidden_layers([30, 20, 10]).
 4
 5  /* Learning paramenters */
 6  max_epochs(30). max_epochs(50).
 7  batch_size(32). batch_size(16).
 8  learning_rate(0.01). learning_rate(0.1).
 9  loss(cross_entropy).
10
11  /* Workflow paramenters */
12  target_performance(0.90). test_percentage(0.2).
13
14  /* Generates all hyper & learning params combinations */
15  params(
16    [hidden_layers(H)],
17    [iterations(X), learning_rate(Y), batch_size(Z), loss(L)]
18  ) :- hidden_layers(H), max_epochs(X), learning_rate(Y),
19       batch_size(Z), loss(L).
20
21  /* Generates and trains all Predictors for the given Dataset and Schema,
22  whose Performance is at least target_performance. */
23  model_selection(Dataset, Schema, Predictor, Transform, Performance) :-
24       test_percentage(R), target_performance(T),
25       random_split(Dataset, R, TrainSet, TestSet),
26       preprocessing_pipeline(TrainSet, Schema, Transform),
27       transform(TrainSet, Transform, ProcessedTrainSet),
28       params(HyperParams, LearnParams),
29       train_cv(ProcessedTrainSet, HyperParams, LearnParams, P),
30       P >= T,
31       multi_layer_perceptron(4, HyperParams, NN),
32       train(NN, TrainingSet, LearnParams, Predictor),
33       transform(TrainSet, Transform, ProcessedTestSet),
34       test(NN, ProcessedTestSet, Performance).
35
36  /* Example of training query: */
37  ?- iris_dataset(D), iris_schema(S), model_selection(D, S, P, _, A).
```

Listing 5: Declarative description of a ML workflow aimed at selecting the best hyper and learning parameters for a NN classifier. Ancillary predicates invoked in this snippet are reported in the supplementary material.

Under these hypotheses, a model selection workflow for the Iris dataset may be triggered via a concise logic query such as the one from listing 5 (line 37). If all aspects of the model selection workflow are correctly declared, the query provide multiple successful solutions corresponding to all trained predictors (`P`) and their test-set accuracies (`A`).

Hybrid reasoning. Finally, listing 6 shows the exploitation of a trained NN predictor as a predicate aimed at classifying (possibly) unseen instances of the Iris flower. The script serves a

```
 1  /* assumption: */
 2  :- iris_dataset(D), iris_schema(S), model_selection(D, S, N, T, _), !,
 3      assert(iris_nn(N, T)).
 4
 5  /* hybrid iris classifier */
 6  iris(SL, SW, PL, PW, Species) :-
 7     iris_nn(Network, Transformation),
 8     transform([SL, SW, PL, PW] , Transformation, ActualX),
 9     predict(Network, ActualX, Y),
10     classify(Y, argmax, [setosa, versicolor, virginica], Species).
```

Listing 6: Exploitation of the NN classifier trained in listing 5 to create an hybrid predicate

```
iris(SL, SW, PL, PW, setosa) :- PW =< 0.78.
iris(SL, SW, PL, PW, versicolor) :- PL >= 2.86, PL < 4.91.
iris(SL, SW, PL, PW, virginica).
```

Listing 7: A purely symbolic classifier for Iris flowers, functionally equivalent to the hybrid one from listing 6

twofold purpose: it exemplifies the ML-Lib functionalities aimed at drawing predictions out of trained ML predictors, and, in particular, it provides an example of an *hybrid* reasoner—where symbolic and sub-symbolic AI seamlessly interoperate.

The script assumes a fact of the form `iris_nn(N, T)` is available into the solver's KB, storing a reference to a trained NN predictor (`N`) and to the affine transformation (`T`) to be applied to each datum the predictor should be fed with. Such assumption may be satisfied, in Prolog, by a query such as the one from listing 6 (line 2) which selects and trains a single NN and stores it into the solver's dynamic KB.

Under such assumption, logic programmers may write an `iris/5` predicate such as the one shown in listing 6. The predicate allows the caller to classify Iris instances by triggering a previously trained NN, and by letting it draw predictions on the data row attained by composing the predicate's arguments—via the `predict/3` predicate. The prediction is then converted into a class constant – via the `classify/4` predicate –, which is in turn bound to the output parameter of `iris/5`—namely `Species`.

It is worth to be highlighted that, from the caller perspective, the `iris/5` described so far is indistinguishable from a purely symbolic predicate serving the same purpose (i.e., Iris classification) and having the same name and arity—such as the one described in listing 7.

6. Conclusions

In this paper, we propose a logic API supporting the seamless integration of logic solvers with sub-symbolic AI, and, in particular neural-network-based supervised ML. Stemming from a domain analysis aimed at identifying the major computational entities involved in a supervised ML workflow, we design our API in terms of computational entities and the operations/functionalities they should support. We then reify our API into a set of logic predicates composing the ML-Lib—i.e., an abstract logic library that any goal-oriented solver may support, there including Prolog ones. Both the syntax and the semantics of each predicate are discussed, as well as architectural and technological requirements. Finally, we provide a number of usage examples aimed at showing the potential of the ML-Lib. In particular, we discuss examples where our logic API supports *declarative* ML (possibly from symbolic data sources), model selection via resolution, and hybrid reasoning. Indeed, the ML-Lib enables the user to formally define ML workflows in a way that is both human- and machine-interpretable, focussing on what should be done, rather than how.

Hybrid reasoning, in particular, is the most relevant contribution of ours. It consists of the seamless integration of logic and sub-symbolic AI at the functional level. In fact, thanks to our ML-Lib, trained sub-symbolic predictors may be used in LP as ordinary predicates.

In the future, we expect contributions to stem from our ML-Lib along two different research threads. The first thread concerns the exploitation of the ML-Lib to create hybrid systems, where LP and ML are integrated into manifold ways. This is made possible by our logic API for ML, which reduces the abstraction gap among LP and ML, as well as the ML-Lib, which lowers the technological barriers preventing the integration of symbolic and sub-symbolic AI. The second thread concerns the extensions of the ML-Lib, which should be eventually delivered to cover currently unsupported functionalities—as well as other ML predictors than NN.

Acknowledgments

This paper has been partially supported by *(i)* the European Union's Horizon 2020 research and innovation programme under G.A. no. 101017142 (StairwAI project), and by *(ii)* the CHIST-ERA IV project CHIST-ERA-19-XAI-005, co-funded by the EU and the Italian MUR (Ministry for University and Research).

References

[1] E. Ilkou, M. Koutraki, Symbolic vs sub-symbolic ai methods: Friends or enemies?, in: CIKM (Workshops), 2020.

[2] R. Calegari, G. Ciatto, A. Omicini, On the integration of symbolic and sub-symbolic techniques for XAI: A survey, Intelligenza Artificiale 14 (2020) 7–32. doi:10.3233/IA-190036.

[3] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI, Information Fusion 58 (2020) 82–115. doi:10.1016/j.inffus.2019.12.012. arXiv:1910.10045.

[4] B. Goertzel, Perception processing for general intelligence: Bridging the symbolic/subsymbolic gap, in: J. Bach, B. Goertzel, M. Iklé (Eds.), Artificial General Intelligence, Springer Berlin Heidelberg, 2012, pp. 79–88.

[5] R. Calegari, G. Ciatto, S. Mariani, E. Denti, A. Omicini, LPaaS as micro-intelligence: Enhancing IoT with symbolic reasoning, Big Data and Cognitive Computing 2 (2018). doi:10.3390/bdcc2030023.

[6] R. Calegari, G. Ciatto, J. Dellaluce, A. Omicini, Interpretable narrative explanation for ML predictors with LP: A case study for XAI, in: F. Bergenti, S. Monica (Eds.), WOA 2019 – 20th Workshop "From Objects to Agents", volume 2404 of *CEUR Workshop Proceedings*, Sun SITE Central Europe, RWTH Aachen University, 2019, pp. 105–112. URL: http://ceur-ws.org/Vol-2404/paper16.pdf.

[7] G. Ciatto, R. Calegari, A. Omicini, 2P-Kt: A logic-based ecosystem for symbolic AI, SoftwareX 16 (2021) 100817:1–7. doi:10.1016/j.softx.2021.100817.

[8] P. Körner, M. Beuschel, J. Barbosa, V. S. Costa, V. Dahl, M. V. Hermenegildo, J. F. Morales, J. Wielemaker, D. Diaz, S. Abreu, G. Ciatto, Fifty years of Prolog and beyond, Theory and Practice of Logic Programming (2022) 1–83. doi:10.1017/S1471068422000102.

[9] Konduit, Deeplearning4J: Open-source distributed deep learning for the JVM, https://deeplearning4j.konduit.ai, 2022. (Supported by the Eclipse Foundation).

[10] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL: http://tensorflow.org/.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[12] F. Chollet, et al., Keras, https://keras.io, 2015.

[13] R. A. Fisher, Iris data set, https://archive.ics.uci.edu/ml/datasets/iris, 1936. (From the UCI Machine Learning Repository).

A. Supplementary Material

A.1. Realising the API: ML-Lib Design

In the reminder of this section, we adopt the following notation to denote the interfaces of logic predicates:

$$\texttt{functor(}\odot_1\ \texttt{Name}_1\texttt{:}\ \textit{type}_1\texttt{, ... ,}\ \odot_N\ \texttt{Name}_N\texttt{:}\ \textit{type}_N\texttt{)}$$

where N denotes the arity of predicate `functor/`N, whose i^{th} argument – named $\texttt{Name}_i$ – must be of type $\textit{type}_i$, and it must be considered as an input or output parameter depending on the mode indicator[1] $\odot_i$. So, for instance, we denote input parameters by `+`, output parameters by `-`, and input-output parameters by `?`. Admissible arguments types include constant term types (`integer`, `real`, `atom`), structured term types (`compound`, `list`), as well as *references* (`ref`), and union types ($\texttt{T}_1\,|\,\texttt{T}_2\,|\ldots$). References, in particular, are a special kind of constant term, whose instances represent objects from the object-oriented realm. These are necessary to make our ML-Lib able to operate with the non-logic entities exposed by the underlying OO library supporting ML.

Accordingly, in the reminder of this section, we enumerate the predicates constituting our ML-Lib, categorised w.r.t. the entities they act upon. In particular, the ML-Lib exposes predicates covering 4 major sorts of entities – i.e. the ones elicited in section 3.2, namely: Schema, Dataset, Transformation, and Predictor –, plus a number of ancillary entities aimed at supporting their manipulation – such as Classification Strategy, Source Type, and Parameter – or specialising their behaviour—such as Neural Network, and Layer.

A.1.1. Schemas

Schemas are concise metadata describing datasets' columns. They define their indexes, names, and admissible types, and they are assumed to be declared by the user.

The ML-Lib supports schemas represented as any of two forms: either as clauses or as objects—to be represented in LP via reference terms. Ad-hoc predicates are provided to support the conversion from one form to the other.

Schemas as clauses. In the general case, schema declarations are firstly provided by the user in clausal form. This requires the user to fill the logic theory with clauses of the form:

$$\begin{array}{l}
\texttt{attribute(1, } N_1\texttt{, } T_1\texttt{).}\\
\quad\vdots\\
\texttt{attribute(}i\texttt{, } N_i\texttt{, } T_i\texttt{).}\\
\quad\vdots\\
\texttt{attribute(}n\texttt{, } N_n\texttt{, } T_n\texttt{).}\\
\texttt{schema\_name(}N\texttt{).}\\
\texttt{schema\_targets([}N_j\texttt{, } N_k\texttt{, } \ldots\texttt{, } N_h\texttt{]).}
\end{array}$$

[1] cf.https://www.swi-prolog.org/pldoc/man?section=preddesc

where N is the name of the schema, and n is the total amount of attributes declared for that schema, while N_i is the name of the i^{th} attribute, and T_i is its type. Indexes $j, k, h \in \{1, \ldots, n\}$ aim at selecting attributes names declared as *targets*—i.e. as outputs of the learning process. While attribute (N_i) and schema (N) names are simple atoms, attribute types (T_i) are compound terms for which the `attribute_type(`T_i`)` holds true.

The `attribute_type/1` predicate is defined as follows:

```
attribute_type(string).
attribute_type(integer).
attribute_type(real).
attribute_type(boolean).
attribute_type(categorical([_ | _])).
attribute_type(ordinal([_ | _])).
```

Hence, admissible attribute types involve infinite domains such as the numeric (either integer or real numbers), and strings ones, as well as finite domains such as booleans, and categorical (i.e. unordered) or ordinal sets of constant values.

Schemas as objects. To be exploitable by the underlying OO library, schemas must be represented as objects. Schemas represented in clausal form can be converted into object form via the following predicate:

```
theory_to_schema(-Schema: ref)
```

which *(i)* inspects the current KB looking for a schema description in clausal form, *(ii)* instantiates a new schema object in the underlying OO library, *(iii)* creates a new reference term referencing the newly created schema, *(iv)* unifies that term with the output parameter denoted by `Schema`.

References to schemas in object form may be then passed as arguments to many other predicates from the ML-Lib in order to provide them the necessary metadata to manipulate datasets.

Manipulating schemas. A part from schema declaration or creation, other relevant operations over schemas involve the inspection (i.e. reading) of their components—namely, names, attribute names, attribute types, and targets. This can be achieved via the following predicate:

```
schema(?Schema: ref, ?Name: atom, ?Attributes: list, ?Targets: list)
```

Given a schema reference, the predicate retrieves *(i)* the schema's name, which is unified with `Name`, *(ii)* the list schema attributes – where each attribute has the form `attribute(`i`, `N_i`, `T_i`)` –, which is unified with `Attributes`, and *(iii)* the list of schema targets – where each target is an atom acting as attribute name –, which is unified with `Targets`. Notably, the predicate is *bi-directional* and its arguments can act as either input or output parameters. In case an unbound `Schema` variable is provided as output parameter, and assuming that the `Name`, `Attributes`, and `Targets` parameters are fully instantiated, the `schema/4` predicate acts as yet another way to create a schema in object form—and the newly created schema is bound to `Schema`.

A.1.2. Datasets

A dataset is a tabular representation of a bunch of homogenous data records. As such, a dataset is characterised by a schema and a number of records matching that schema.

Similarly to what it does for schemas, the ML-Lib supports datasets represented as either clauses or objects. Ad-hoc predicates are provided to support the conversion from one form to the other, other than for loading datasets from some data source, such as a file or a DBMS.

Datasets as objects. In the general case, datasets objects are firstly loaded from a data source. These may be local or remote files – commonly in "comma separated values" (CSV) format –, as well as DBMS of any sort—provided that adequate connection support is provided by the underlying OO library, or any other third-party module. The ML-Lib provides a unique entry point to load a dataset from any data source, namely:

`read_dataset(+Location: *atom*, +SourceType: *atom*, -Dataset: *ref*)`

This predicate aims at loading the dataset from a given `Location`—be it a path on the local filesystem, a URL referencing some remote resource, or a connection string for some DBMS. It also requires the caller to specify the `SourceType` the dataset should be read from. Regardless of the particular location and source type, the behaviour of the `read_dataset/3` predicate is such that: *(i)* raw data is retrieved from `Location`, and *(ii)* parsed according to the selected source `SourceType`; finally *(iii)* a new dataset object is created along with a reference term for it, *(iv)* which is then unified with `Dataset`.

Admissible values for the `SourceType` parameter are determined by the `source_type/1` predicate, defined as follows:

`source_type(*csv*).`

meaning that currently the ML-Lib only supports data provisioning from CSV files. However, further source types are going be supported in the future. That will imply extending the `source_type/1` predicate definition with further cases.

Datasets as clauses. Logic programmers may also be willing to describe the dataset via a logic theory. When this is the case, the theory should contain not only the clauses describing the schema (i.e. the dataset's columns), but also a number of clauses describing the actual content of the dataset (i.e. its rows). In particular, the ML-Lib expects data entries to be provided as clauses of the form:

$$\begin{array}{c} N(\mathtt{X}_{1,1},\ \ldots,\ \mathtt{X}_{1,j},\ \ldots,\ \mathtt{X}_{1,n}). \\ \vdots \\ N(\mathtt{X}_{i,1},\ \ldots,\ \mathtt{X}_{i,j},\ \ldots,\ \mathtt{X}_{i,n}). \\ \vdots \\ N(\mathtt{X}_{m,1},\ \ldots,\ \mathtt{X}_{m,j},\ \ldots,\ \mathtt{X}_{m,n}). \end{array}$$

where N is the schema name declared via `schema_name/1`, and $\texttt{X}_{i,j}$ is the value of the j^{th} attribute of the i^{th} data entry. Of course, the actual type of $\texttt{X}_{i,j}$ must be coherent with the formal type T_i declared in the schema definition.

Datasets in clausal form must be converted into object form to be exploitable by the underlying OO library. This can be achieved via the following predicate:

`theory_to_dataset(+SchemaName:` *`atom`*`, -Dataset:` *`ref`*`)`

which *(i)* inspects the current KB looking for one or clauses using `SchemaName` as the head functor, *(ii)* instantiates a new dataset object in the underlying OO library, *(iii)* populates it with as many rows as the aforementioned clauses, *(iv)* creates a new reference term referencing the newly created dataset, *(v)* unifies that term with the output parameter denoted by `Dataset`. Of course, this predicate also takes into account the schema-related metadata which are assumed to be defined in clausal form as well.

Datasets manipulation. Datasets are amongst the basic bricks of predictors training in ML, hence they must support several kinds of manipulations. Within the scope of the ML-Lib, we support partitioning a dataset in several ways to support both cross validation and test set separation, other than accessing a dataset by row, column, or cell. Conversions from and into clausal form complete the picture.

Splitting. To support test set separation, the ML-Lib provides a predicate to randomly split a dataset into a training and test set, given a ratio:

`random_split(+Dataset:` *`ref`*`, +Ratio:` *`real`*`, -Train:` *`ref`*`, -Test:` *`ref`*`)`

Given a reference to a `Dataset` in object form, and a `Ratio` – i.e. a real number in the range $]0, 1[$ –, the predicate *(i)* randomly samples the given percentage of data entries from `Dataset`, *(ii)* collects them into a new dataset, whose reference is bound to `Test`, and *(iii)* collects the remaining data entries into yet another dataset, whose reference is bound to `Train`. So, for instance, a ratio of 0.1 would randomly split the dataset into a training set containing 90% of the original data, and a test set containing 10% of the original data.

To support cross validation, ML-Lib provides an *ad-hoc* predicate:

`fold(+Dataset:` *`ref`*`, +K:` *`integer`*`, -Train:` *`ref`*`, -Validation:` *`ref`*`)`

which splits the `Dataset` into 2 partitions, namely `Train` and `Validation`, the former containing $\frac{k-1}{k}$% data entries – to be used as the training set –, and the latter containing the remaining $\frac{1}{k}$% data entries—to be used as the validation set. Both `Train` and `Validation` are bound to reference terms, referencing datasets in object form. Notably, the `fold/2` is non-deterministic as it enumerates all possible folds of a `K`-fold cross validation process. Hence, provided that $\texttt{K} \geq 2$, the predicate computes `K` partitioning of the original dataset.

Data access. The ML-Lib supports accessing the information encapsulated into a dataset in object form via three predicates, namely:

- `row(+Dataset:` *`ref`*`, ?Index:` *`integer`*`, -Values:` *`list`*`).`
- `column(+Dataset:` *`ref`*`, ?Attribute:` *`integer`*`|`*`atom`*`, -Values:` *`list`*`).`

- `cell(+Dataset:` *`ref`*, `?Index:` *`integer`*, `?Attribute:` *`integer`*|*`atom`*, `-Values:` *`list`*`)`.

They are all non-deterministic, and they both support the retrieval of a particular row / column / cell from the dataset as well as the enumeration of all possible rows / columns / cells from that dataset.

In particular, predicate `row/3` aims at retrieving rows. If the `Index` parameter is a positive integer, then the predicate attempts to unify the `Value` parameter with the list of values contained the `Index`th row of the dataset. Otherwise, if `Index` is uninstantiated, the predicate enumerates all rows in the dataset, and for each row it unifies the `Index` and `Values` parameters accordingly.

The predicate `column/3` is totally analogous to `row/3`, expect it aims at retrieving or enumerating columns. The only notable difference w.r.t. `row/3` is that columns can be referenced by either attribute names or indexes—thus both positive integers and atoms can be bound to the `Attribute` parameter.

Finally, predicate `cell/4` supports accessing or enumerating cells. In particular, it allows the user to access the `Value` in position (`Index`, `Attribute`), where `Index` is a row index in and `Attribute` is an attribute name or index. If one or both parameters are uninstantiated, the predicate enumerates all possible assignments.

Object to clausal form conversion. The logic programmer may also be willing to convert a dataset in object form into a dataset in clausal form. This can be attained via the following predicate:

`theory_from_dataset(+Schema:` *`ref`*, `+Dataset:` *`ref`*`)`

Given the references to both a dataset and its schema in object form, the predicate populates the solver's *dynamic* KB with the a number of clauses representing the dataset and its schema in the clausal form described above.

A.1.3. Transformations

A transformation is a function altering a dataset and, possibly, its schema. It may be parametric and hence tuned according to the content of the dataset or its schema.

Consider for instance the case of the "Normalization" transformation. It applies an affine transformation to each column of the dataset (independently) in such a way that it has a predefined mean (e.g. 0) and standard deviation (e.g. 1). Hence, it alters the content of a dataset leaving its schema unaffected. To work properly, it requires two major computational steps, namely *(i)* computing (and storing) the mean and standard deviation of each column of the original dataset, *(ii)* applying the affine transformation to normalize the dataset columns (i.e. subtracting the mean and dividing by the standard deviation each cell of each column).

In the general case, transformations are modelled as *stateful* entities supporting at least 2 operations, namely *fitting* and *transforming* a dataset and its schema. The latter operation is also known as "applying a transformation to a dataset", and it should not only support the retrieval of the transformed dataset, but the transformed schema as well. Furthermore, transformations should be composable into *pipelines*, i.e. cascades of simpler transformations to be fitted or applied in a row.

To support all such aspects, the ML-Lib provides predicates aiming to

1. create a transformation given a schema,
2. combine elementary transformations into composite transformations,
3. fit transformations over data (regardless of whether they are elementary or composite),
4. apply composite or elementary transformation to a dataset, thus attaining a new dataset,
5. retrieve the new schema resulting from a transformation application.

Differently from schemas and datasets, for which the ML-Lib supports both clausal and object representations, transformations are only representable in object form, hence the following predicates assume transformations to be manipulated via reference terms.

Transformations to/from schemas. To support aims 1 and 5, the ML-Lib provides the following *bi-directional* predicate:

```
schema_transformation(?Schema: ref, ?Transformation: ref)
```

which changes its behaviour depending on which arguments are instantiated.

In particular, if `Schema` is bound to a schema object, then `Transformation` is unified with an identity transformation – i.e. a transformation leaving the schema and the dataset unaffected –, which can be used as the initial step of a composite pipeline. This is how aim 1 is served.

Conversely, if `Transformation` is bound to an actual transformation object, then `Schema` is unified with the new schema object attained by applying that transformation to the schema it was originally constructed from. This is how aim 5 is served.

Creating and combining elementary transformations. To support aim 2, the ML-Lib provides a number of predicates sharing a similar syntax. Each predicate is in charge of creating a composite transformation by appending a specific elementary transformation to some previously created one—like, for instance, the identity transformation created via `schema_transformation/2`.

In the general case, the combination and creation of transformations is attained via predicates of the form:

$\langle name \rangle$`(+Pipeline`$_{in}$`: ref, +`A_1`, . . . , +`A_n`, -Pipeline`$_{out}$`: ref)`

where $\langle name \rangle$ is the name of the transformation being appended to `Pipeline`$_{in}$, while $A_1, \ldots, A_n$ are transformation-specific parameters, and `Pipeline`$_{out}$ is the output parameter to which the newly created transformation is bound.

The ML-Lib currently supports 3 predicates of this sort, and further ones may be defined following the same syntactical convention. These are `normalize/3`, `one_hot_encoding/3`, and `attributes_delete/3`, and their details are described later in this paragraph. Here we focus on the overall design which is aimed at supporting the declaration of *pipelines* of transformations, via conjunctions of goals:

```
theory_to_schema(OriginalSchema),
schema_transformation(OriginalSchema,T0),
transformation1(T0, arg1, T1),
    ⋮
transformationm(Tm−1, argm, Tm),
schema_transformation(FinalSchema, Tm)
```

Following this convention, logic programmers may declaratively construct the pipeline of transformations to be applied to `OriginalSchema` to produce `FinalSchema`, in such a way that each variable $\mathtt{T}_i$, for $i \in \{0, \ldots, m\}$ is bound to an object summarising all transformation steps from 0 to i.

Normalization. A dataset's columns can be normalised in such a way that, for each column, the mean is 0 and the standard deviation is 1. Such kind of transformations may alter the dataset while leaving its schema unaffected. A normalization transformation can be created via the following predicate:

normalize(+Pipeline$_{in}$: *ref*, +Attributes: *list|atom*, -Pipeline$_{out}$: *ref*)

There, parameter `Attributes` must be bound to either a list of attribute names or indexes – denoting the columns to be normalized –, or the '`all`' atom—denoting a situation where all columns should be normalized.

One Hot Encoding. A dataset's target attributes whose type are categorical with k-admissible values can be replaced by k binary attributes, via one-hot encoding (OHE) transformations. Such kind of transformations alter both the dataset and its schema. A OHE transformation can be created via the following predicate:

one_hot_encode(+Pipeline$_{in}$: *ref*, +Attributes: *list|atom*, -Pipeline$_{out}$: *ref*)

There, parameter `Attributes` must be bound to a list of attribute names or indexes denoting the columns to be one-hot encoded.

Attributes Deletion. Columns may be dropped from a dataset and its schema via attribute deletion transformations. Such kind of transformations alter both the dataset and its schema. An attribute deletion transformation can be created via the following predicate:

one_hot_encode(+Pipeline$_{in}$: *ref*, +Attributes: *list|atom*, -Pipeline$_{out}$: *ref*)

There, parameter `Attributes` must be bound to a list of attribute names or indexes denoting the columns to be dropped.

Fitting transformations to data. To support aim 3, the ML-Lib provides the following predicate:

fit(+Transformation$_{in}$: *ref*, +Dataset: *ref*, -Transformation$_{out}$: *ref*) which works by tuning $\texttt{Transformation}_{in}$ over `Dataset`, producing a new transformation, whose reference is unified with $\texttt{Transformation}_{out}$.

The new transformation may be identical to the input one, in case the latter does not require tuning—such as in the case of OHE. Conversely, in case it does need tuning – as in the case of normalization –, the output transformation may actually be different than the original one. Fitting a composite transformation of course has the effect of fitting all its components, recursively.

Applying transformations to data. Finally, to support aim 4, the ML-Lib provides the following *bi-directional* predicate:

$$\texttt{transform(?Data}_{in}\texttt{: } \mathit{ref}\,|\,\mathit{compound}\texttt{, +Transformation: } \mathit{ref}\texttt{, ?Data}_{out}\texttt{: } \mathit{ref}\,|\,\mathit{compound}\texttt{)}$$

which can either apply a transformation or its inverse depending on either entire datasets or their rows, depending on how arguments are passed.

In particular, $\texttt{Data}_{in}$ and $\texttt{Data}_{out}$ can be either dataset references, or compound terms, denoting single rows. Of course, applying a (possibly inverse) transformation to a row (resp. entire dataset) shall produce a row (resp. entire dataset) in return.

The predicate applies `Transformation` to $\texttt{Data}_{in}$ in case the latter parameter is instantiated, unifying the transformed result with $\texttt{Data}_{out}$. Conversely, it applies the inverse of `Transformation` to $\texttt{Data}_{out}$ in case the $\texttt{Data}_{in}$ parameter is uninstantiated while the former is not. When this is the case, the transformed result is unified with $\texttt{Data}_{in}$.

A.1.4. Predictors

Predictors are stateful entities which can be *trained* over a dataset to later draw *predictions* on new data matching the same schema. In the general case, all predictors may require a number of *hyper parameters* to be specified upon creation, and a number or *learning parameters* to be provided upon training. Both kinds of parameters aim at regulating the predictor behaviour, either in general or during training, and their actual values must be decided by the user.

Given the large number of possible predictors from the data science literature, the ML-Lib just fixes the syntactical convention to support predictors creation, other than the API to support both training and drawing predictions. Notably, as for transformations, the ML-Lib assumes predictors to be represented in object form, and therefore manipulated via reference terms.

Creating predictors. The ML-Lib constrains predictor-creating predicates to comply to the following syntactical convention:

$$\langle \mathit{name} \rangle\texttt{(+}H_1\texttt{, ... , +}H_n\texttt{, -Predictor: } \mathit{ref}\texttt{)}$$

where $\langle \mathit{name} \rangle$ is the name of the predictor type being instantiated, while $H_1, \ldots, H_n$ are predictor-type-specific hyper-parameters, and `Predictor` is the output parameter to which the newly created predictor is bound.

The ML-Lib currently supports one predicate of this sort – namely, the `neural_network/2` predicate, described later in this section –, yet further ones may be defined following the same syntactical convention.

Training. Regardless of their nature, predictors can be trained on data via the following predicate:

$$\texttt{train(+Predictor}_{in}\texttt{: } \textit{ref}\texttt{, +Dataset: } \textit{ref}\texttt{, +Params: } \textit{list}\texttt{, -Predictor}_{out}\texttt{: } \textit{ref}\texttt{)}$$

The predicate accepts $\texttt{Predictor}_{in}$ as the predictor to be trained, the `Dataset` it should be trained upon, and a list of predictor-specific `Params`. Behind the scenes, the predicate exploits a predictor-specific learning algorithm to train $\texttt{Predictor}_{in}$, possibly following the suggestions/constraints carried by `Params`. Once the training has been completed, a reference to the trained predictor is bound to $\texttt{Predictor}_{out}$, and the execution of the predicate succeeds.

Learning Parameters. The `Params` argument of `train/4` must be instantiated with a list of learning parameters aimed at controlling and constraining the execution of a learning algorithm. In the general case, each parameter is a term of the form:

$$\langle name \rangle \texttt{(} \langle value \rangle \texttt{)}$$

where $\langle name \rangle$ is a functor describing the purpose of the parameter, while $\langle value \rangle$ is an arbitrary term acting as value for the parameter.

In the particular case of neural networks, the ML-Lib admits the following learning parameters

- `max_epochs(N: ` *`integer`*`)` limiting the amount of epochs[2] to be performed while training a NN;
- `batch_size(N: ` *`integer`*`)` defining the amount of training samples to be taken into account in each single step of the learning algorithm;
- `learning_rate(R: ` *`real`*`)` defining the step size in a gradient descent learning process;
- `loss(Function: ` *`atom`*`)` dictating which loss function should be optimised during training (admissible values include: `mse` for mean squared error, `mae` for mean absolute error, `cross_entropy`, etc.)

Other sorts of learning parameters may be added to the ML-Lib, targeting both NN or other sorts of predictors.

Drawing predictions. Regardless of their nature, *trained* predictors can be exploited to draw predictions from data – e.g. from a whole dataset or a single row –, via the following predicate:

$$\texttt{predict(+Predictor: } \textit{ref}\texttt{, +InputData: } \textit{ref} \mid \textit{compound}\texttt{, -Prediction: } \textit{ref} \mid \textit{compound}\texttt{)}$$

[2]i.e., the amount of times the learning algorithm works through the entire training dataset

The predicate accepts a `Predictor` (which must have been previously trained via `train/4`), and some `InputData` – which may either be reference to a dataset object, or a compound term denoting a single row –, and uses the `Predictor` to compute a prediction for each data entry in `InputData`. Predictions may consist of either a single row or a whole dataset, depending on how many data entries are contained in `InputData`. In both cases, the `Prediction` output parameter is unified with the predicted row/dataset.

In case `InputData` is bound to a full dataset including one or more target columns, those target columns are ignored while computing predictions. Conversely, when `InputData` is bound to a list of values, the ML-Lib considers them all as input values.

Classification. As many predictors – there including NN – are technically tailored on *regression* tasks (where predicted values are real numbers), it is a common practice for data scientists to map *classification* tasks (where predicted values are categorical) onto regression tasks, to make it possible to address them via regressors. The mapping commonly works as follows. A classification task requiring input data to be classified according to $k \in \mathbb{N}_{\geq 0}$ classes, can be conceived as a regression aimed at predicting continuos vectors $\mathbf{y} \in \mathbb{R}^k$ from the same input data. Given a particular input datum $\mathbf{x}$, and the corresponding prediction $\mathbf{y}$, the i^{th} component of $\mathbf{y}$ – namely, y_i – could then be interpreted as the confidence of $\mathbf{x}$ being classified as an example of the i^{th} class. Depending on the nature of the classification task at hand, the confidence values in $\mathbf{y}$ could be jointly interpreted following several strategies. In a situation where classes are mutually exclusive, one may use function $argmax_i(y_i)$ to select the most likely class of $\mathbf{x}$. Otherwise, if classes can overlap, one choose a confidence threshold θ and classify $\mathbf{x}$ according to all those classes i such that $y_i \geq \theta$.

The ML-Lib supports classification out of regressors via the following predicate:

```
classify(+Prediction: ref|compound, +Strategy: compound, +Classes:
            list, -Classification: ref|compound)
```

which accepts a `Prediction` computed via `predict/3` – be it a single row or a whole dataset –, a classification `Strategy`, a list of `Classes`, and an output parameter, `Classification`, which is bound to a container for as many categorical predictions as in `Prediction`.

Notably, while the `Classes` parameter must consist of a list of (at least 2) class names, admissible values for the `Strategy` parameter are determined by the `classification/1` predicate, defined as follows:

```
classification(argmax).
classification(threshold(Th)) :- numeric(Th).
```

meaning that currently the ML-Lib only supports classification via the `argmax` or threshold-based strategies—despite further strategies may be added following the same syntactical notation.

Assessing Predictions. Predictors can be assessed by comparing their *actual* predictions with a test dataset containing *expected* predictions, having no overlap with the data used during training. Several scoring functions can be used to serve this purpose, like, for instance mean squared/absolute error (MSE/MAE) or R^2 for regressors, as well as accuracy, recall, or F1-Score for classifiers.

The ML-Lib supports assessing a predictor via a number of predicates following the same syntactical convention:

⟨*name*⟩`(+Actual:` *`ref`*`|`*`list`*`, +Expected:` *`ref`*`|`*`list`*`, -Score:` *`real`*`)`

where ⟨*name*⟩ is the name of the scoring function of choice, `Actual` is either a dataset or a list containing the actual predictions produced by the predictor under assessment, `Actual` is either a dataset or a list containing the test data, and `Score` is the output parameter to be unified with the score value computed whenever the predicate is executed.

Notable cases of scoring functions are, for instance: `mse/3`, `mae/3`, `r2/3`, `accuracy/3`, `recall/3`, or `f1_score/3`, while further ones may be added following the same syntactical convention.

A.1.5. Neural Networks

Neural networks are a particular sort of predictor. They consist of directed acyclic graphs (a.k.a. DAG) where vertices are elementary computational units called neurons, and edges (a.k.a. synapses) are weighted.

Topologically, neural networks are organised in layers, and data scientists design them by specifying *(i)* how many layers compose the network, *(ii)* how many neurons compose each layer, *(iii)* which activation function is used by each layer – and therefore by each neuron therein contained –, and *(iv)* how are layers – and therefore their neurons – interconnected with their predecessors and successors in the DAG. Hence, a NN's hyper-parameters should provide information about such aspects.

The ML-Lib provides the following predicate to construct NN-like predictors:

`neural_network(+Topology:` *`ref`*`, -Predictor:` *`ref`*`)`

There, `Topology` is a reference to an object describing the overall architecture of the network, and, in particular its layers.

Layers. Layered architectures are commonly composed by at least one input layer – whose neurons simply mirror the input data –, and one output layer—whose neurons' output values jointly represent the NN prediction. In the between an arbitrary amount of layers of different sorts may be defined—e.g. dense, convolutional, pooling, etc. In all such cases, declaring a layer implies specifying its sort, size (in terms of neurons), and activation function.

The ML-Lib supports the declaration of layered architectures similarly to how it supports pipelines of transformations. There are two major sorts of predicates to serve this purpose:

- `input_layer(+Size:` *`integer`*`, -Layer:` *`ref`*`).`
- ⟨*type*⟩`_layer(+Previous:` *`ref`*`, +Size:` *`integer`*`, +Activation: ref, -Layer:` *`ref`*`).`

The former predicate, `input_layer/2`, aims at creating a `Layer` of a given `Size`. The size should match the amount of input attributes in the training dataset. This is the entry point of any cascade of predicates aimed at creating a layered architecture.

Conversely, the latter predicate pattern, $\langle type \rangle$`_layer/4` is matched by a number of actual predicates aimed at creating intermediate or output layers. There $\langle type \rangle$ denotes the type of the layer. Regardless of their type, these predicates accept a reference to some `Previous` layer, whose output synapses are connected to the layer under construction, in a way which depends by its type. They also accept the `Size` of the layer to be constructed, and the `Activation` function its neurons should employ. Finally, they all accept an output parameter, `Layer`, to which a reference to the newly created layer is bound, in case creation succeeds.

The `dense_layer/4` predicate is a notable case matching the aforementioned pattern. It aims at declaring a layer whose neurons are *densely* connected with its predecessor's ones—in the sense that, each neuron of the predecessor has an outgoing synapsis towards each neuron of the dense layer. Layers of such a sort are commonly exploited as intermediate. Conversely, layers declared via the `output_layer/4` predicate – again matching the aforementioned pattern – are commonly *final* in any well formed NN architecture.

So, for instance, an ordinary multi-layered perceptron (MLP) composed by 1 input layer with 4 neurons, 1 hidden layer with 7 neurons, and 1 output layer with 3 neurons, where all neurons exploit the sigmoid activation function, can be declared as follows:

```
input_layer(4, I),
dense_layer(I, 7, sigmoid, H),
output_layer(H, 3, sigmoid, O),
neural_network(O, NN)
```

There variable `I` is bound to the input layer, variable `H` is bound to the hidden layer, and `O` is bound to the output layer, whereas `NN` is bound to a MLP predictor whose architecture comprehends `I`, `H`, and `O`.

Activation Functions. The behaviour of neurons should be finely tuned via their activation function. Indeed, all layer-creating predicates of the form $\langle type \rangle$`_layer/4` expect an activation function to be provided by the user. Admissible activation functions are regulated by the `activation/1` predicate, defined below:

activation(*identity*). denoting $f(x) = x$
activation(*sigmoid*). denoting $f(x) = 1/(1 + e^{-x})$
activation(*tanh*). denoting $f(x) = tanh(x)$
activation(*relu*). denoting $f(x) = max(0, x)$

while others may be possibly added.

B. Model selection: further details

The model selection example discussed in section 5 and formally described in listing 5 relies upon a number of ancillary predicates declaring some particular steps of the workflow and exemplifying many ML-Lib functionalities. These are reported in listing 8. For instance, `train_cv/4` is in charge of performing 10-fold CV on a given `Dataset`, to assess a given `HyperParams-LearnParams` combination, to then compute the `AveragePerformance` of the 10 predictors constructed in this way. Every single fold of a `K`-fold CV process is managed

```prolog
/* Trains a NN multiple times, over Dataset, using the provided Params. */
/* Returns the AveragePerformance over a 10-fold CV. */
train_cv(Dataset, HyperParams, LearnParams, AveragePerformance) :-
    findall(
        Performance,
        train_cv_fold(Dataset, 10, HyperParams, LearnParams, Performance),
        AllPerformances
    ),
    mean(AllPerformances, AveragePerformance).

/* Trains a NN once, for the k-th round of CV. */
/* Returns the Performance over the k-th validation set. */
train_cv_fold(Dataset, K, HyperParams, LearnParams, Performance) :-
    fold(Dataset, K, Train, Validation),
    train_validate(Train, Validation, HyperParams, LearnParams, Performance).

/* Tranis a NN on the provided TrainingSet, using the provided Params, */
/* and computes its Performance over the provided ValidationSet. */
train_validate(TrainingSet, ValidationSet, HyperParams, LearnParams, Performance) :-
    multi_layer_perceptron(4, HyperParams, 3, NN),
    train(NN, TrainingSet, LearnParams, TrainedNN),
    test(NN, ValidationSet, Performance).

% Computes the Performance of the provided NN against the provided ValidationSet
test(NN, ValidationSet, Performance) :-
    predict(NN, ValidationSet, ActualPredictions),
    accuracy(ActualPredictions, ValidationSet, Performance).
```

Listing 8: Ancillary predicates used in listing 5. Each predicate denotes one particular step of a model selection workflow

by the `train_cv_fold/5` predicate, which in turn exploits `train_validate/5` predicate to train and validate every single predictor. Finally, the `test/3` predicate can be exploited to either test or validate a predictor depending on whether the test or validation set is provided as an argument.

Temporalizing Epistemic Logic L-DINF[⋆]

Stefania Costantini[1,3], Andrea Formisano[2,3,*] and Valentina Pitoni[1]

[1]*DISIM - Università dell'Aquila, via Vetoio–loc. Coppito, 67100 L'Aquila, Italy*
[2]*DMIF - Università di Udine, via delle Scienze 206, 33100 Udine, Italy*
[3]*GNCS-INdAM, piazzale Aldo Moro 5, 00185 Roma, Italy*

Abstract
Agents and Multi-Agent Systems (MAS) are a technology that has many fields of application, which extend also to human sciences and where Computational Logic has been widely applied. In this paper, we join together two of our long-lasting lines of work in this field. In particular, we introduce time and time intervals into the epistemic logic *L-DINF*, that copes with group dynamics in MAS.

Keywords
Epistemic Logic Programs, Agents and Multi-Agent Systems, Temporal Reasoning

1. Introduction

Agents and Multi-Agent Systems are a technology that has many fields of application, which extend also to human sciences (cf., e.g., the recent book [1]). The applications of Computational Logic in the field of agents and MAS are several, as can be seen in the surveys [2, 3] (the latter being very recent). Logic is, in fact, often used to model such kind of systems, as it (at least potentially) provides verifiability and explainability. In this paper, we join together two of our long-lasting lines of work in this field.

The first one [4, 5, 6] was aimed at introducing a treatment of time in agents, so that, upon reception of new perceptions that led to acquire new beliefs, the agent would not have to override old beliefs, but rather to update the time interval where they resulted to hold. In order not to restrict the application of our approach only to certain agent-oriented frameworks, we defined in [6] a "time module" suitable to add time in an easy way into many logic representations of agents. This module is in practice a particular kind of function, that we called T, that assigns a "timing" to atoms, in terms of either single instants or time intervals. We drew inspiration for this work from methods to design agent memorization mechanisms inspired, in turn, by models of human memory [7, 8], that have been developed in cognitive science.

The second line of work [9, 10, 11] has been aimed to formally model via epistemic logic (aspects of) the group dynamics of cooperative agents. Our overall objective has been to devise

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
[⋆] Research partially supported by Action COST CA17124 "DigForASP" and by projects INDAM GNCS-2020 *NoRMA* and INDAM GNCS-2022 InSANE (CUP_E55F22000270001).
[*] Corresponding author.
stefania.costantini@univaq.it (S. Costantini); andrea.formisano@uniud.it (A. Formisano); valentina.pitoni@univaq.it (V. Pitoni)
0000-0002-5686-6124 (S. Costantini); 0000-0002-6755-9314 (A. Formisano); 0000-0002-4245-4073 (V. Pitoni)

agent-oriented logical frameworks that allows a designer to formalize and formally verify Multi-Agent Systems, modelling the capability to construct and execute joint plans within a group of agents. We devote a special attention on explainability, in the perspective of Trustworty AI: to enable human-level explanations to be generated, the syntax of our logic is especially devised to make it possible to transpose a proof into a natural language. All along, we have taken into particular account the connection between theory and practice, so as to make our logic actually usable by a system's designers. So, we care about aspects related to enabling and performing physical actions, and to agent's memory of the action performed, where those aspects are often neglected in related work. We have proposed in particular a framework called the Logic of "Inferable" *L-DINF*, based on epistemic logic, where a group of cooperative agents can jointly perform actions. I.e., at least one agent of the group can perform the action, either with the approval of the group or on behalf of the group. We have taken into consideration actions' *cost* [10], and the preferences that each agent can have for what concerns performing each action [11]. We have recently introduced agents' *roles* within a group, in terms of the actions that each agent is enabled by its group to perform. *L-DINF* has a fully-defined semantics, and a proof of strong completeness w.r.t. canonical models. In this paper we incorporated the T function into *L-DINF*, so that actions, goals, plans, of groups of agents are now temporalized, and thus refer to time instants or time intervals. We have joined the two semantic approaches, preserving the strong completeness of the axiomatic framework.

The paper is organized as follows. In Section 2 we present syntax, an example of application to a simple planning problem, namely, a group of researches co-authoring a paper to be submitted, and semantics of the enhanced epistemic logic. Section 3 presents a revised definition of canonical model to take into account the temporal aspect. Finally, in Section 4 we shortly conclude.

2. Logical Framework

L-DINF is a logic which consists of a static component and a dynamic one. The static component, called *L-INF*, is a logic of explicit beliefs and background knowledge. The dynamic component, called *L-DINF*, extends the static one with dynamic operators capturing the consequences of the agents' inferential actions on their explicit beliefs as well as a dynamic operator capturing what an agent can conclude by performing some inferential action in its repertoire.

2.1. Syntax

Let be $Atm = \{p(t_1, t_2), q(t_3, t_4), \ldots, h(t_i, t_j), \ldots\}$ where p, q, h are predicate symbols and each $t_\ell \in \mathbb{N}$. Here an atomic proposition of the form $p(t_1, t_2)$ stands for *"p is true from the time instant t_1 to t_2"* with $t_1 \leqslant t_2$ (*Temporal Representation* of the external world); as a special case we can have $p(t_1, t_1)$ which stands for *"p is true in the time instant t_1"*. We also admit predicate symbols of higher arity, but in that case we assume that the first two arguments are those that identify the time duration of the belief (e.g., the atomic proposition $open(1, 3, \text{door})$ means "the agent knows that the door is open from time 1 to time 3"). By $Prop$ we denote the set of all propositional formulas, i.e. the set of all Boolean formulas built out of the set of atomic propositions Atm. The set Atm_A represents the physical actions that an agent can

perform, including “active sensing” actions (e.g., “let’s check whether it rains”, “let’s measure the temperature”). Let Agt be a set of agents. In what follows, I is a MTL “time-interval” [12] which is a closed finite interval $[t, l]$ or an infinite interval $[t, \infty)$ (considered open on the upper bound), for any expressions/values t, l such that $0 \leq t \leq l$.
The language of *L-DINF*, denoted by $\mathcal{L}_{\textit{L-DINF}}$, is defined by the following grammar:

$$\begin{array}{lll}
\varphi, \psi & ::= & p(t_1, t_2) \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathbf{B}_i\, \varphi \mid \mathbf{K}_i\, \varphi \mid \Box_I\, \varphi \mid do_i(\phi_A, I) \mid can\_do_i(\phi_A, I) \mid \\
& & do_G(\phi_A, I) \mid can\_do_G(\phi_A, I) \mid pref\_do_i(\phi_A, d, I) \mid pref\_do_G(i, \phi_A, I) \mid \\
& & exec_i(\alpha) \mid exec_G(\alpha) \mid [G : \alpha]\, \varphi \mid intend_i(\phi_A, I) \mid intend_G(\phi_A, I) \\
\alpha & ::= & \vdash\!(\varphi,\psi) \mid \cap(\varphi,\psi) \mid \downarrow\!(\varphi, \psi) \mid \dashv\!(\varphi, \psi)
\end{array}$$

where $p(t_1, t_2)$ ranges over Atm, $d \in \mathbb{N}$, $i \in Agt$ and $G \subseteq Agt$. (Other Boolean operators are defined from $\neg$ and $\wedge$ in the standard manner.) The language of *inferential actions* of type α is denoted by $\mathcal{L}_{\mathsf{ACT}}$. The static part *L-INF* of *L-DINF*, includes only those formulas not having sub-formulas of type α.

Notice the expression $intend_i(\phi_A, I)$, where it is required that $\phi_A \in Atm_A$ and I is a time interval. This expression indicates the intention of agent i to perform action ϕ_A in the interval I in the sense of the BDI agent model [13]. This intention can be part of an agent’s knowledge base from the beginning, or it can be derived later. In this paper we do not cope with the formalization of BDI, for which the reader may refer, e.g., to [14]. So, we will treat intentions rather informally, assuming also that $intend_G(\phi_A, I)$ holds whenever all agents in group G intend to perform action ϕ_A in the interval I.

The formula $do_i(\phi_A, I)$, indicates *actual execution* of action ϕ_A by agent i. By precise choice, do (and similarly do_G, that indicates the actual execution of ϕ_A by the group of agents G) are not axiomatized. In fact, they are realized by what has been called in [15] a *semantic attachment*, i.e., a procedure which connects an agent with its external environment in a way that is unknown at the logical level. The axiomatization concerns only the relationship between doing and being enabled to do.

The expressions $can\_do_i(\phi_A, I)$ and $pref\_do_i(\phi_A, d, I)$ (where, as before, $\phi_A \in Atm_A$ and I is a time interval) are closely related to $do_i(\phi_A, I)$. In fact, $can\_do_i(\phi_A, I)$ is to be seen as an enabling condition, indicating that agent i is enabled to execute action ϕ_A in the interval I, while instead $pref\_do_i(\phi_A, d, I)$ indicates the level d of preference/willingness of agent i to perform that action in the time interval I. $pref\_do_G(i, \phi_A, I)$ indicates that agent i exhibits the *maximum level* of preference on performing action ϕ_A within all group members in the time interval I. Notice that, if a group of agents intends to perform an action ϕ_A, this will entail that the entire group intends to do ϕ_A, that will be enabled to be actually executed only if at least one agent $i \in G$ can do it, i.e., it can derive $can\_do_i(\phi_A, I)$.

Unlike explicit beliefs, i.e., facts and rules acquired via perceptions during an agent’s operation and kept in the *working memory*, an agent’s background knowledge is assumed to satisfy *omniscience* principles, such as closure under conjunction and known implication, and closure under logical consequence, and introspection. In fact, $\mathbf{K}_i$ is actually the well-known S5 modal operator often used to model/represent knowledge. The fact that background knowledge is closed under logical consequence is justified because we conceive it as a kind of stable reliable *knowledge base*, or *long-term memory*. We assume the background knowledge to include: facts (formulas) known by the agent from the beginning, and facts the agent has later decided to store in its long-term memory (by means of some decision mechanism not treated here) after having processed them in its working memory. We therefore assume background knowledge to be irrevocable, in the sense of being stable over time.

In the formula $\Box_I\,\phi$ the MTL Interval "always" operator is applied to a formula, which means that ϕ is always true in the interval I. $\Box_{[0,\infty)}$ will sometimes be written simply as $\Box$.

A formula of the form $[G{:}\alpha]\,\varphi$, with $G \subseteq Agt$, and where α must be an inferential action, states that "φ holds after action α has been performed by at least one of the agents in G, and all agents in G have common knowledge about this fact".

Borrowing from [11, 16], we distinguish four types of inferential actions α which allow us to capture some of the dynamic properties of explicit beliefs and background knowledge: $\downarrow(\varphi,\psi)$, $\cap(\varphi,\psi)$, $\dashv(\varphi,\psi)$, and $\vdash(\varphi,\psi)$, These actions characterize the basic operations of forming explicit beliefs via inference:

- $\downarrow(\varphi,\psi)$: this action infers ψ from φ, where ψ is an atom, say $p(t_1,t_2)$: an agent, believing that φ is true and having in its long-term memory that φ implies ψ (in some suitable time interval including $[t_1,t_2]$), starts believing that $p(t_1,t_2)$ is true.
- $\cap(\varphi,\psi)$: this action closes the explicit beliefs φ and ψ under conjunction. I.e., $\cap(\varphi,\psi)$ characterizes the inferential action of deducing $\varphi \wedge \psi$ from the explicit belief φ and the explicit belief ψ.
- $\dashv(\varphi,\psi)$: this action performs a simple form of "belief revision", where φ and ψ are atoms, say $p(t_1,t_2)$ and $q(t_3,t_4)$ respectively: an agent, believing $p(t_1,t_2)$ and having in the long-term memory that $p(t_1,t_2)$ implies $\neg q(t_3,t_4)$, removes the timed belief $q(t_3,t_4)$ if the intervals match. Notice that, should q be believed in a wider interval I such that $[t_1,t_2] \subseteq I$, the belief $q(.,.)$ is removed concerning intervals $[t_1,t_2]$ and $[t_3,t_4]$, but it is left for the remaining sub-intervals (so, its is "restructured").
- $\vdash(\varphi,\psi)$: let ψ be an atom, say $p(t_1,t_2)$. An agent, believing φ and that φ implies $p(t_1,t_2)$ in the working memory (in some suitable time interval including $[t_1,t_2]$), starts believing $p(t_1,t_2)$. This last action operates directly on the working memory without retrieving anything from the background knowledge.

Formulas of the forms $exec_i(\alpha)$ and $exec_G(\alpha)$ express executability of inferential actions either by agent i, or by a group G of agents (which is a consequence of any of the group members being able to execute the action). It has to be read as: "α is an inferential action that agent i (resp. an agent in G) can perform".

2.2. Problem Specification and Inference: An Example

In this section, we propose an example to explain the usefulness of this kind of logic and to help the reader's understanding. Consider a group G of three agents, who are the authors of a paper that has to be submitted to a conference: the first author a deals with the drafting of the introduction and finding the references, the second b deals with the experiments and the third c deals with the formalization part. The second is the only one who can perform the experiments because he has the required certifications; the others are enabled to perform different tasks, such as, e.g., write the abstract, search references, check the correctness of the formal part, and so on.

The group receives notification of a deadline for a paper, so they decide to organize themselves for submitting it. The group will reason, and devise the intention/goal $\mathbf{K}_i(\Box_I\, intend_G(submit\_fullpaper(t_0, t_2), I))$: the group intends to submit their paper within the indicated time I. Here t_0 is the time instant when the group begins to organize to write the paper, $I = [t_0, t_1]$ where t_1 is the deadline and t_2 is the time instant when they really submit the paper and $t_2 \leq t_1$.

Among the physical actions that agents in the group can perform are for instance the following: $submit\_abstract$, $do\_experiment$, $write\_introduction$,$write\_formal\_part$ and $write\_experiment\_results$.

The group will now be required to perform a planning activity. Assume that, as a result of the planning phase, the knowledge base of each agent i contains the following rule, that specifies how to reach the intended goal in terms of actions to perform and sub-goals to achieve (listed after the " $\rightarrow$ "):

$$\begin{aligned}\mathbf{K}_i\big(\Box_I\, intend_G(submit\_fullpaper(t_0, t_2), I) \rightarrow \Box_{I_1}\, intend_G(submit\_abstract(t_0, t_3), I_1)\\ \wedge \Box_{I_2}\, intend_G(do\_experiment(t_0, t_4), I_2)\\ \wedge \Box_I\, intend_G(write\_formal\_part(t_0, t_5), I)\big)\end{aligned}$$

where $I_1, I_2 \subseteq I$, t_3 is the time instant when the author submit the abstract and $t_3 \leq t_1$, t_4 is the time instant when the author b has finished his experiment and he has written the results at $t_4 \leq t_1$, finally t_5 is the time instant when the other agent has finished to write the formal part. Thanks to the axiomatization, which we are going to explain in Section 2.5, we have that $intend_G(\phi_A, I) \leftrightarrow \forall i \in G\; intend_i(\phi_A, I)$, each agent has the specialized rule (for $i \leq 3$):

$$\begin{aligned}\mathbf{K}_i\big(\Box_I\, intend_i(submit\_fullpaper(t_0, t_2), I) \rightarrow \Box_{I_1}\, intend_i(submit\_abstract(t_0, t_3), I_1) \wedge\\ \Box_{I_2}\, intend_i(do\_experiment(t_0, t_4), I_2) \wedge\\ \Box_I\, intend_i(write\_formal\_part(t_0, t_5), I)\big)\end{aligned}$$

Therefore, the following is entailed for each of the agents:

$$\begin{aligned}&\mathbf{K}_i\big(\Box_I\, intend_i(submit\_fullpaper(t_0, t_2), I) \rightarrow \Box_{I_1}\, intend_i(submit\_abstract(t_0, t_3), I_1)\big)\\ &\mathbf{K}_i\big(\Box_I\, intend_i(submit\_fullpaper(t_0, t_2), I) \rightarrow \Box_{I_2}\, intend_i(do\_experiment(t_0, t_4), I_2)\big)\\ &\mathbf{K}_i\big(\Box_I\, intend_i(submit\_fullpaper(t_0, t_2), I) \rightarrow \Box_I\, intend_i(write\_formal\_part(t_0, t_5), I)\big).\end{aligned}$$

Assume now that the knowledge base of each agent i contains also the following general rules, stating that the group is available to perform each of the necessary actions. Which agent will in particular perform each action ϕ_A? According to items (t4) and (t7) in the definition of truth values, listed in the next section, for *L-DINF* formulas, this agent will be chosen as the one which best prefers to perform this action, among those that can do it. Formally, in the present situation, $pref\_do_G(i, \phi_A, I)$ identifies the agent i in the group with the highest degree of preference on performing ϕ_A, and $can\_do_G(\phi_A, I)$ is true if there is some agent i in the group which is able and allowed to perform ϕ_A, i.e., $\phi_A \in A(i, w) \wedge \phi_A \in H(i, w)$.

$$\mathbf{K}_i\big(\Box_{I_1}(intend_G(submit\_abstract(t_0,t_3),I_1) \wedge can\_do_G(submit\_abstract(t_0,t_3),I_1) \wedge pref\_do_G(i, submit\_abstract(t_0,t_3),I_1)) \rightarrow \Box_{I_1} do_G(submit\_abstract(t_0,t_3),I_1)\big)$$
$$\mathbf{K}_i\big(\Box_{I_2}(intend_G(do\_experiment(t_0,t_4),I_2) \wedge can\_do_G(do\_experiment(t_0,t_4),I_2) \wedge pref\_do_G(i, do\_experiment(t_0,t_4),I_2)) \rightarrow \Box_{I_2} do_G(do\_experiment(t_0,t_4),I_2)\big)$$
$$\mathbf{K}_i\big(\Box_{I}(intend_G(write\_formal\_part(t_0,t_5),I) \wedge can\_do_G(write\_formal\_part(t_0,t_5),I) \wedge pref\_do_G(i, write\_formal\_part(t_0,t_5),I)) \rightarrow \Box_{I} do_G(write\_formal\_part(t_0,t_5),I)\big)$$

As before, such rules can be specialized to each single agent.

$$\mathbf{K}_i\big(\Box_{I_1}(intend_i(submit\_abstract(t_0,t_3),I_1) \wedge can\_do_i(submit\_abstract(t_0,t_3),I_1) \wedge pref\_do_i(i, submit\_abstract(t_0,t_3),I_1)) \rightarrow \Box_{I_1} do_i(submit\_abstract(t_0,t_3),I_1)\big)$$
$$\mathbf{K}_i\big(\Box_{I_2}(intend_i(do\_experiment(t_0,t_4),I_2) \wedge can\_do_i(do\_experiment(t_0,t_4),I_2) \wedge pref\_do_i(i, do\_experiment(t_0,t_4),I_2)) \rightarrow \Box_{I_2} do_i(do\_experiment(t_0,t_4),I_2)\big)$$
$$\mathbf{K}_i\big(\Box_{I}(intend_i(write\_formal\_part(t_0,t_5),I) \wedge can\_do_i(write\_formal\_part(t_0,t_5),I) \wedge pref\_do_i(i, write\_formal\_part(t_0,t_5),I)) \rightarrow \Box_{I} do_i(write\_formal\_part(t_0,t_5),I)\big)$$

So, for each action ϕ_A required by the plan, there will be some agent (let us assume for simplicity only one), for which $do_i(\phi_A, I)$ will be concluded. In our case, the agent a will conclude $do_a(submit\_abstract(t_0,t_3),I_1)$; the agent b will conclude $do_b(do\_experiment(t_0,t_4),I_2)$ and the agent c will conclude $do_c(write\_formal\_part(t_0,t_5),I)$.

2.3. Semantics

Now we can go into the details of semantics, definition 2.1 introduces the notion of *L-INF model*, which is then used to introduce semantics of the static fragment of the logic. Before that we define the "time" function T that associates to each formula the time interval in which this formula is true and operates as follows:

- $T(p(t_1,t_2)) = [t_1,t_2]$, which stands for *"p is true in the time interval* $[t_1,t_2]$*"* where $t_1,t_2 \in \mathbb{N}$; as a special case we have $T(p(t_1,t_1)) = t_1$, which stands for *"p is true in the time instant* t_1*"* where $t_1 \in \mathbb{N}$ (time instant);
- $T(\neg p(t_1,t_2)) = T(p(t_1,t_2))$, which stands for *"p is not true in the time interval* $[t_1,t_2]$*"* where $t_1,t_2 \in \mathbb{N}$;
- $T(\varphi \text{ op } \psi) = T(\varphi) \uplus T(\psi)$ with $op \in \{\vee, \wedge, \rightarrow\}$, which is the unique smallest interval including both $T(\varphi)$ and $T(\psi)$;
- $T(\mathbf{B}_i\varphi) = T(\varphi)$;
- $T(\mathbf{K}_i\varphi) = T(\varphi)$;
- $T(\Box_I\varphi) = I$ where I is a time interval;
- $T([(G:\alpha)]\varphi)$ there are different cases depending on the inferential action α:
  1. $T([G : \downarrow(\varphi,\psi)]\,\psi) = T(\psi)$;
  2. $T([G : \cap(\varphi,\psi)]\,(\varphi \wedge \psi)) = T(\varphi) \uplus T(\psi)$, the smallest interval including $T(\varphi)$ and $T(\psi)$;
  3. $T([G : \dashv(\varphi,\psi)]\,\psi)$ returns the "restructured" interval where ψ is true;
  4. $T([G : \vdash(\varphi,\psi)]\,\psi) = T(\psi)$;

- $T(do_i(\phi_A, I)) = T(do_G(\phi_A, I)) = I$;
- $T(can\_do_i(\phi_A, I)) = T(can\_do_G(\phi_A, I)) = I$;
- $T(intend_i(\phi_A, I)) = T(intend_G(\phi_A, I)) = I$;
- $T(pref\_do_i(\phi_A, d, I)) = T(pref\_do_G(i, \phi_A, I)) = I$;
- $T(exec_i(\alpha)) = T(exec_G(\alpha)) = T([(G : \alpha)]\varphi)$.

Definition 2.1, below, depends on a given *set of world* W and a *valuation function*, namely a mapping $V : W \longrightarrow 2^{Atm}$. For each world $w \in W$, let t_1 the minimum time instant of $T(p(t_1, t))$ where $p(t_1, t) \in V(w)$ and let t_2 be the supremum time instant (we can have $t_2 = \infty$) w.r.t. the atoms $p(t, t_2)$ in $V(w)$. Whenever useful, we denote w as w_I where $I = [t_1, t_2]$, which identifies the world in a given interval.

Notice that many relevant aspects of an agent's behaviour are specified in the definition of *L-INF model*, including which mental and physical actions an agent can perform, which is the cost of an action and which is the budget that the agent has available, which is the preference degree of the agent to perform each action. This choice has the advantages of keeping the complexity of the logic under control, and of making these aspects modularly modifiable. As before let Agt be the set of agents.

Definition 2.1. *A model is a tuple $M = (W, N, \mathcal{R}, E, B, C, A, H, P, V, T)$ where:*

- *W is a set of worlds (or situations);*
- *$\mathcal{R} = \{R_i\}_{i \in Agt}$ is a collection of equivalence relations on W: $R_i \subseteq W \times W$ for each $i \in Agt$;*
- *$N : Agt \times W \longrightarrow 2^{2^W}$ is a neighborhood function such that, for each $i \in Agt$, each $w_I, v_I \in W$, and each $X \subseteq W$ these conditions hold:*

  (C1) *if $X \in N(i, w_I)$ then $X \subseteq \{v_I \in W \mid w_I R_i v_I\}$,*
  (C2) *if $w_I R_i v_I$ then $N(i, w_I) = N(i, v_I)$;*

- *$E : Agt \times W \longrightarrow 2^{\mathcal{L}_{\mathsf{ACT}}}$ is an executability function of mental actions such that, for each $i \in Agt$ and $w_I, v_I \in W$, it holds that:*

  (D1) *if $w_I R_i v_I$ then $E(i, w_I) = E(i, v_I)$;*

- *$B : Agt \times W \longrightarrow \mathbb{N}$ is a budget function such that, for each $i \in Agt$ and $w_I, v_I \in W$, the following holds*

  (E1) *if $w_I R_i v_I$ then $B(i, w_I) = B(i, v_I)$;*

- *$C : Agt \times \mathcal{L}_{\mathsf{ACT}} \times W \longrightarrow \mathbb{N}$ is a cost function such that, for each $i \in Agt$, $\alpha \in \mathcal{L}_{\mathsf{ACT}}$, and $w_I, v_I \in W$, it holds that:*

  (F1) *if $w_I R_i v_I$ then $C(i, \alpha, w_I) = C(i, \alpha, v_I)$;*

- *$A : Agt \times W \longrightarrow 2^{Atm_A}$ is an executability function for physical actions such that, for each $i \in Agt$ and $w_I, v_I \in W$, it holds that:*

  (G1) *if $w_I R_i v_I$ then $A(i, w_I) = A(i, v_I)$;*

- *$H : Agt \times W \longrightarrow 2^{Atm_A}$ is an enabling function for physical actions such that, for each $i \in Agt$ and $w_I, v_I \in W$, it holds that:*

(**G2**) *if* $w_I R_i v_I$ *then* $H(i, w_I) = H(i, v_I)$*;*

- $P : Agt \times W \times Atm_A \longrightarrow \mathbb{N}$ *is a preference function for physical actions* ϕ_A *such that, for each* $i \in Agt$ *and* $w_I, v_I \in W$*, it holds that:*

  (**H1**) *if* $w_I R_i v_I$ *then* $P(i, w_I, \phi_A) = P(i, v_I, \phi_A)$*;*
- $V : W \longrightarrow 2^{Atm}$ *is a valuation function;*
- *T is the "Time Function", defined before.*

To simplify the notation, let $R_i(w_I) = \{v_I \in W \mid w_I R_i v_I\}$, for $w_I \in W$. The set $R_i(w_I)$ identifies the situations that agent i considers possible at world w_I. It is the *epistemic state* of agent i at w_I. In cognitive terms, $R_i(w_I)$ can be conceived as the set of all situations that agent i *can retrieve* from its long-term memory and reason about.

While $R_i(w_I)$ concerns background knowledge, $N(i, w_I)$ is the set of all facts that agent i explicitly believes at world w_I, a fact being identified with a set of worlds. Hence, if $X \in N(i, w_I)$ then, the agent i has the fact X under the focus of its attention and believes it. We say that $N(i, w_I)$ is the explicit *belief set* of agent i at world w_I.

The executability of inferential actions is determined by the function E. For an agent i, $E(i, w_I)$ is the set of inferential actions that agent i can execute at world w_I in time interval I. The value $B(i, w_I)$ is the budget the agent has available to perform inferential actions in time interval I. Similarly, the value $C(i, \alpha, w_I)$ is the cost to be paid by agent i to execute the inferential action α in the world w_I in time interval I. The executability of physical actions is determined by the function A. For an agent i, $A(i, w_I)$ is the set of physical actions that agent i can execute at world w_I in time interval I. $H(i, w_I)$ instead is the set of physical actions that agent i is enabled by its group to perform always in I. Which means, H defines the *role* of an agent in its group, via the actions that it is allowed to execute.

Agent's preference on executability of physical actions is determined by the function P. For an agent i, and a physical action ϕ_A, $P(i, w_I, \phi_A)$ is an integer value d indicating the degree of willingness of i to execute ϕ_A at world w_I.

Constraint (**C1**) imposes that agent i can have explicit in its mind only facts which are compatible with its current epistemic state. Moreover, according to constraint (**C2**), if a world v_I is compatible with the epistemic state of agent i at world w_I, then agent i should have the same explicit beliefs at w_I and v_I. In other words, if two situations are equivalent as concerns background knowledge, then they cannot be distinguished through the explicit belief set. This aspect of the semantics can be extended in future work to allow agents make plausible assumptions. Analogous properties are imposed by constraints (**D1**), (**E1**), and (**F1**). Namely, (**D1**) imposes that agent i always knows which inferential actions it can perform and those it cannot. (**E1**) states that agent i always knows the available budget in a world (potentially needed to perform actions). (**F1**) determines that agent i always knows how much it costs to perform an inferential action. (**G1**) and (**H1**) determine that an agent i always knows which physical actions it can perform and those it cannot, and with which degree of willingness, where (**G2**) specifies that an agent also knows whether its group gives it the permission to execute a certain action or not, i.e., if that action pertains to its *role* in the group.

Given a model $M = (W, N, \mathcal{R}, E, B, C, A, H, P, V, T)$, $i \in Agt$, $G \subseteq Agt$, $w_I \in W$, and a formula $\varphi \in \mathcal{L}_{\textit{L-INF}}$, we introduce the following shorthand notation:

$$\|\varphi\|^M_{i,w_I} = \{v_I \in W : w_I R_i v_I \text{ and } M, v_I \models \varphi\}$$

whenever $M, v_I \models \varphi$ is well-defined (see below). Then, truth values of *L-DINF* formulas are inductively defined as follows:

(t1) $M, w_I \models p(t_1, t_2)$ iff $p(t_1, t_2) \in V(w_I)$ and $T(p(t_1, t_2)) \subseteq I$
(t2) $M, w_I \models exec_i(\alpha)$ iff $\alpha \in E(i, w_I)$ and $T(exec_i(\alpha)) \subseteq I$
(t3) $M, w_I \models exec_G(\alpha)$ iff $\exists i \in G$ with $\alpha \in E(i, w_I)$ and $T(exec_G(\alpha)) \subseteq I$
(t4) $M, w_I \models can\_do_i(\phi_A, J)$ iff $\phi_A \in A(i, w_I) \cap H(i, w_I)$ and $J \subseteq I$
(t5) $M, w_I \models can\_do_G(\phi_A, J)$ iff $\exists i \in G$ with $\phi_A \in A(i, w_I) \cap H(i, w_I)$ and $J \subseteq I$
(t6) $M, w_I \models pref\_do_i(\phi_A, d, J)$ iff $\phi_A \in A(i, w_I)$, $P(i, w_I, \phi_A) = d$ and $J \subseteq I$
(t7) $M, w_I \models pref\_do_G(i, \phi_A, J)$ iff $M, w \models pref\_do_i(\phi_A, d, J)$ for $d = \max\{P(j, w, \phi_A) \mid j \in G \wedge \phi_A \in A(j, w) \cap H(j, w)\}$ and $J \subseteq I$
(t8) $M, w_I \models \neg\varphi$ iff $M, w \not\models \varphi$ and $T(\neg\varphi) \subseteq I$
(t9) $M, w_I \models \varphi \wedge \psi$ iff $M, w \models \varphi$ and $M, w \models \psi$ with $T(\varphi), T(\psi) \subseteq I$
(t10) $M, w_I \models \mathbf{B}_i\, \varphi$ iff $||\varphi||^M_{i,w} \in N(i, w)$ with $T(\varphi) \subseteq I$
(t11) $M, w_I \models \mathbf{K}_i\, \varphi$ iff $M, v \models \varphi$ for all $v \in R_i(w)$ with $T(\varphi) \subseteq I$
(t12) $M, w_I \models \Box_J\, \varphi$ iff $T(\varphi) \subseteq J \subseteq I$ and for all $v_I \in R_i(w_I)$ it holds $M, w_I \models \varphi$

As seen above, a physical action can be performed by a group of agents if at least one agent of the group can do it, and the level of preference for performing this action is set to the maximum among those of the agents enabled to do this action. For any inferential action α performed by any agent i, we set:
$M, w \models [G : \alpha]\varphi$ iff $M^{[G:\alpha]}, w \models \varphi$
where $M^{[G:\alpha]} = \langle W, N^{[G:\alpha]}, \mathcal{R}, E, B^{[G:\alpha]}, C, A, H, P, V, T\rangle$, is the model representing the fact that the execution of an inferential action α affects the sets of beliefs of agent i and modifies the available budget in a certain time interval I. Such operation can add new beliefs by direct perception, by means of one inference step, or as a conjunction of previous beliefs. Hence, when introducing new beliefs (i.e., performing mental actions), the neighborhood must be extended accordingly.

The following condition characterizes the circumstances in which an action may be performed, and by which agent(s):

$$enabled_{w_I}(G, \alpha) : \ \exists j \in G\, (\alpha \in E(j, w) \wedge \tfrac{C(j,\alpha,w_I)}{|G|} \leq \min_{h \in G} B(h, w_I))$$

with $T([G{:}\alpha]\varphi) \subseteq I$. This condition states when an inferential action is enabled. In the above particular formulation (that is not fixed, but can be customized to the specific application domain) if at least an agent can perform it and if the "payment" due by each agent (obtained by dividing the action's cost equally among all agents of the group) is within each agent's available budget. In case more than one agent in G can execute an action, we implicitly assume the agent j performing the action to be the one corresponding to the lowest possible cost. Namely, j is such that $C(j, \alpha, w_I) = \min_{h \in G} C(h, \alpha, w_I)$. Other choices might be viable, so variations of this logic can be easily defined simply by devising some other enabling condition and, possibly, introducing differences in neighborhood update. Notice that the definition of the enabling function basically specifies the "concrete responsibility" that agents take while concurring with their own resources to actions' execution. Also, in case of specification of various resources, different corresponding enabling functions might be defined.

2.4. Belief Update

In this kind of logic, updating an agent's beliefs accounts to modify the neighborhood of the present world. The updated neighborhood $N^{[G:\alpha]}$ resulting from execution of a mental action α by a group G of agents is as follows.

$$N^{[G:\downarrow(\psi,\chi)]}(i,w_I) = \begin{cases} N(i,w_I) \cup \{||\chi||^M_{i,w_I}\} & \text{if } i \in G \text{ and } T([G:\downarrow(\psi,\chi)]\chi) \subseteq I \text{ and} \\ & \mathit{enabled}_{w_I}(G,\downarrow(\psi,\chi)) \text{ and} \\ & M, w_I \models \mathbf{B}_i\psi \wedge \mathbf{K}_i(\psi \to \chi) \\ N(i,w_I) & \text{otherwise} \end{cases}$$

$$N^{[G:\cap(\psi,\chi)]}(i,w_I) = \begin{cases} N(i,w_I) \cup \{||\psi \wedge \chi||^M_{i,w_I}\} & \text{if } i \in G \text{ and} \\ & T([G:\cap(\psi,\chi)](\psi \wedge \chi)) \subseteq I \text{ and} \\ & \mathit{enabled}_{w_I}(G,\cap(\psi,\chi)) \text{ and} \\ & M, w_I \models \mathbf{B}_i\psi \wedge \mathbf{B}_i\chi \\ N(i,w_I) & \text{otherwise} \end{cases}$$

$$N^{[G:\vdash(\psi,\chi)]}(i,w_I) = \begin{cases} N(i,w_I) \cup \{||\chi||^M_{i,w_I}\} & \text{if } i \in G \text{ and } T([G:\dashv(\psi,\chi)]\chi) \subseteq I \text{ and} \\ & \mathit{enabled}_{w_I}(G,\vdash(\psi,\chi)) \text{ and} \\ & M, w_I \models \mathbf{B}_i\psi \wedge \mathbf{B}_i(\psi \to \chi) \\ N(i,w_I) & \text{otherwise} \end{cases}$$

Notice that, after an inferential action α has been performed by an agent $j \in G$, all agents $i \in G$ see the same update in the neighborhood. Conversely, for any agent $h \notin G$ the neighborhood remains unchanged (i.e., $N^{[G:\alpha]}(h,w) = N(h,w_I)$). However, even for agents in G, the neighborhood remains unchanged if the required preconditions, on explicit beliefs, knowledge, and budget, do not hold (and hence the action is not executed). Notice also that we might devise variations of the logic by making different decisions about neighborhood update to implement, for instance, partial visibility within a group.

For formulas of the form $[G : \dashv(\psi,\chi)]\chi$, we take in account the following ground case: given $Q = q(j,k)$ such that $T(q(j,k)) = T(q(t_1,t_2)) \cap T(q(t_3,t_4))$ with $j,k \in \mathbb{N}$ and $P \equiv \Big(\big(M, w_I \models \mathbf{B}_i(p(t_1,t_2)) \wedge \mathbf{B}_i(q(t_3,t_4)) \wedge \mathbf{K}_i(p(t_1,t_2) \to \neg q(t_3,t_4))\big)$ and $\big(T([G :\dashv (p(t_1,t_2), q(t_3,t_4))]q(t_5,t_6)) \subseteq I\big)$ and there is no interval $J \supsetneq T(p(t_1,t_2))$ s.t. $\mathbf{B}_i(q(t_5,t_6))$

where $T(q(t_5, t_6)){=}J\Big)$:

$$N^{[G:\dashv(p(t_1,t_2),q(t_3,t_4))]}(i, w_I) = \begin{cases} N(i, w_I) \setminus \{||Q||^M_{i,w_I}\} & \text{if } P \text{ holds} \\ N(i, w_I) & \text{otherwise} \end{cases}$$

The following update of the budget function determines how each agent in G contributes to cover the costs of execution of an action, by consuming part of its available budget. We assume, however, that only inferential actions that add new beliefs have a cost. I.e., forming conjunction and performing belief revision are actions with no cost. As before, for an action α, we require $\mathit{enabled}_{w_I}(G, \alpha)$ to hold and assume that $j \in G$ executes α. Then, depending on α, we have:

$$B^{[G:\downarrow(\psi,\chi)]}(i, w_I) = \begin{cases} B(i, w_I) - \frac{C(j,\downarrow(\psi,\chi),w_I)}{|G|} & \text{if } i \in G \text{ and } T([G : \downarrow(\psi, \chi)]\chi) \subseteq I \text{ and} \\ & \mathit{enabled}_{w_I}(G, \downarrow(\psi, \chi)) \text{ and} \\ & M, w_I \models \mathbf{B_I}_i\psi \wedge \mathbf{K}_i(\psi \to \chi) \\ B(i, w_I) & \text{otherwise} \end{cases}$$

$$B^{[G:\vdash(\psi,\chi)]}(i, w_I) = \begin{cases} B(i, w_I) - \frac{C(j,\vdash(\psi,\chi),w_I)}{|G|} & \text{if } i \in G \text{ and } T([G : \vdash(\psi,\chi)]\chi) \subseteq I \text{ and} \\ & \mathit{enabled}_{w_I}(G, \vdash(\psi,\chi)) \text{ and} \\ & M, w_I \models \mathbf{B}_i\psi \wedge \mathbf{B}_i(\psi \to \chi) \\ B(i, w_I) & \text{otherwise} \end{cases}$$

We write $\models_{\mathit{L\text{-}DINF}} \varphi$ to denote that $M, w_I \models \varphi$ holds for all worlds w_I of every model M.

We introduce below relevant consequences of our formalization. For lack of space we omit the proof, that can be developed analogously to what done in previous work [10]. For any set of agents G and each $i \in G$, we have the following:

- $\models_{\mathit{L\text{-}INF}} (\mathbf{K}_i(\varphi \to \psi)) \wedge \mathbf{B}_i\, \varphi) \to [G : \downarrow(\varphi, \psi)]\, \mathbf{B}_i\, \psi$.
  Namely, if an agent has φ among beliefs and $\mathbf{K}_i(\varphi \to \psi)$ in its background knowledge, then as a consequence of the action $\downarrow(\varphi, \psi)$ the agent and any group G to which it belongs start believing ψ.
- $\models_{\mathit{L\text{-}INF}} (\mathbf{K}(p(t_1, t_2) \to \neg q(t_3, t_4)) \wedge \mathbf{B}_i p(t_1, t_2) \wedge \mathbf{B}_i q(t_3, t_4)) \to$
  $[(G : \dashv(p(t_1, t_2), q(t_3, t_4)))]\mathbf{B}_i q(t_5, t_6)$,
  where $T(q(t_5, t_6)) = T(q(t_3, t_4)) \setminus T(q(t_1, t_2))$.
  Namely, if agent i has $q(t_3, t_4)$ as one of its beliefs, q is not believed outside $T(q(t_3, t_4))$, the agent perceives $p(t_1, t_2)$ where $T(p(t_1, t_2)) \subseteq T(q(t_3, t_4))$, and has $\mathbf{K}_i(p(t_1, t_2) \to \neg q(t_3, t_4))$ in its background knowledge. Then after the mental operation $\dashv(p(t_1, t_2), q(t_3, t_4))$ the agent starts believing $q(t_5, t_6))$ where $T(q(t_5, t_6)) = T(q(t_3, t_4)) \setminus T(q(t_1, t_2))$.
- $\models_{\mathit{L\text{-}INF}} (\mathbf{B}_i\varphi \wedge \mathbf{B}_i\psi) \to [G : \cap(\varphi, \psi)]\mathbf{B}_i(\varphi \wedge \psi)$.
  Namely, if an agent has φ and ψ as beliefs, then as a consequence of the action $\cap(\varphi, \psi)$ the agent and any group G to which it belongs start believing $\varphi \wedge \psi$.

- $\models_{\textit{L-INF}} (\mathbf{B}_i(\varphi \to \psi)) \wedge \mathbf{B}_i\,\varphi) \to [G : \vdash(\varphi,\psi)]\,\mathbf{B}_i, \psi$.
  Namely, if an agent has φ among its beliefs and $\mathbf{B}_i(\varphi \to \psi)$ in its working memory, then as a consequence of the action $\vdash(\varphi,\psi)$ the agent and any group G to which it belongs start believing ψ.

2.5. Axiomatization

Below we introduce the axiomatization of our logic. The *L-INF* and *L-DINF* axioms and inference rules are the following:

1. $(\mathbf{K}_i\,\varphi \wedge \mathbf{K}_i(\varphi \to \psi)) \to \mathbf{K}_i\,\psi$;
2. $\mathbf{K}_i\,\varphi \to \varphi$;
3. $\neg\mathbf{K}_i(\varphi \wedge \neg\varphi)$;
4. $\mathbf{K}_i\,\varphi \to \mathbf{K}_i\,\mathbf{K}_i\,\varphi$;
5. $\neg\mathbf{K}_i\,\varphi \to \mathbf{K}_i\,\neg\mathbf{K}_i\,\varphi$;
6. $\mathbf{B}_i\,\varphi \wedge \mathbf{K}_i\,(\varphi \leftrightarrow \psi) \to \mathbf{B}_i\,\psi$;
7. $\mathbf{B}_i\,\varphi \to \mathbf{K}_i\,\mathbf{B}_i\,\varphi$;
8. $\Box_I\varphi \wedge \Box_I(\varphi \to \psi) \to \Box_I(\psi)$;
9. $\Box_I\varphi \to \Box_J\varphi$ with $J \subseteq I$;
10. $\frac{\varphi}{\mathbf{K}_i\,\varphi}$;
11. $[G : \alpha]p \leftrightarrow p$;
12. $[G : \alpha]\neg\varphi \leftrightarrow \neg[G : \alpha]\varphi$;
13. $exec_G(\alpha) \to \mathbf{K}_i\,(exec_G(\alpha))$;
14. $[G : \alpha](\varphi \wedge \psi) \leftrightarrow [G : \alpha]\varphi \wedge [G : \alpha]\psi$;
15. $[G : \alpha]\mathbf{K}_i\,\varphi \leftrightarrow \mathbf{K}_i\,([G : \alpha]\varphi)$;
16. $[G : \downarrow(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \mathbf{B}_i\,([G : \downarrow(\varphi,\psi)]\chi) \vee [G : \downarrow(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \big((\mathbf{B}_i\,\varphi \wedge \mathbf{K}_i\,(\varphi \to \psi)) \wedge [G : \downarrow(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \mathbf{K}_i\,([G : \downarrow(\varphi,\psi)]\chi \leftrightarrow \psi)\big)$;
17. $[G : \cap(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \mathbf{B}_i\,([G : \cap(\varphi,\psi)]\chi) \vee [G : \cap(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \big((\mathbf{B}_i\,\varphi \wedge \mathbf{B}_i\,\psi) \wedge [G : \cap(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \mathbf{K}_i\,[G : \cap(\varphi,\psi)]\chi \leftrightarrow (\varphi \wedge \psi)\big)$;
18. $[G : \vdash(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \mathbf{B}_i\,([G : \vdash(\varphi,\psi)]\chi) \vee [G : \vdash(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \big((\mathbf{B}_i\,\varphi \wedge \mathbf{B}_i\,(\varphi \to \psi)) \wedge [G : \vdash(\varphi,\psi)]\mathbf{B}_i\,\chi \leftrightarrow \mathbf{K}_i\,([G : \vdash(\varphi,\psi)]\chi \leftrightarrow \psi)\big)$;
19. $[G : \dashv(\varphi,\psi)]\neg\mathbf{B}_i\,\chi \leftrightarrow \mathbf{B}_i\,([G : \dashv(\varphi,\psi)]\chi) \vee [G : \dashv(\varphi,\psi)]\neg\mathbf{B}_i\,\chi \leftrightarrow \big((\mathbf{B}_i\,\varphi \wedge \mathbf{K}_i\,(\varphi \to \neg\psi)) \wedge [G : \dashv(\varphi,\psi)]\neg\mathbf{B}_i\,\chi \leftrightarrow \mathbf{K}_i\,([G : \dashv(\varphi,\psi)]\chi \leftrightarrow \psi)\big)$;
20. $intend_G(\phi_A, I) \leftrightarrow \forall i \in G\; intend_i(\phi_A; I)$;
21. $do_G(\phi_A, I) \to can\_do_G(\phi_A, I)$;
22. $do_i(\phi_A, I) \to can\_do_i(\phi_A, I) \wedge pref\_do_G(i, \phi_A, I)$;
23. $\frac{\psi \leftrightarrow \chi}{\varphi \leftrightarrow \varphi[\psi/\chi]}$.

We write *L-DINF* $\vdash \varphi$ to denote that φ is a theorem of *L-DINF*. It can be verified that the above axiomatization is sound for the class of *L-INF* models, namely, all axioms are valid and inference rules preserve validity. In particular, soundness of axioms 16–19 follows from the semantics of $[G{:}\alpha]\varphi$, for each inferential action α, as previously defined. Notice that, by abuse of notation, we have axiomatized the special predicates concerning intention and action enabling. Axioms 20–22 concern in fact physical actions, stating that: what is intended by a group of agents is intended by them all; and, neither an agent nor a group of agents can do what it is not enabled to do. Such axioms are not enforced by the semantics, but are supposed to be enforced by a designer's/programmer's encoding of parts of an agent's behaviour. In fact, axiom 20 enforces agents in a group to be cooperative. Axioms 21 and 22 ensure that agents will attempt to perform actions only if their preconditions are satisfied, i.e., if they can do them. We do not handle such properties in the semantics as done, e.g., in dynamic logic, because we want agents' definition to be independent of the practical aspect, so we explicitly intend to introduce flexibility in the definition of such parts.

3. Canonical Model and Strong Completeness

In this section we adapt the notion of canonical model for *L-INF* introduced in [10] to deal with the time component. The proof of strong completeness of the framework directly exploits the notion of canonical model by applying a standard argument. Time is handled in the semantics by means of the time function T and the definition of canonical *L-INF* model is immediately obtained from the one in [10], as follows:

Definition 3.1. *Let* Agt *be a set of agents. The* canonical *L-INF model is a tuple* $M_c = \langle W_c, N_c, \mathcal{R}_c, E_c, B_c, C_c, A_c, H_c, P_c, V_c, T_c \rangle$ *where:*

- W_c *is the set of all maximal consistent subsets of* $\mathcal{L}_{\textit{L-INF}}$*;*
- $\mathcal{R}_c = \{R_{c,i}\}_{i \in Agt}$ *is a collection of equivalence relations on* W_c *such that, for every* $i \in Agt$ *and* $w_I, v_I \in W_c$*,* $w_I R_{c,i} v_I$ *if and only if (for all* φ*,* $\mathbf{K}_i\, \varphi \in w_I$ *implies* $\varphi \in v_I$*);*
- *For* $w \in W_c$*,* $\varphi \in \mathcal{L}_{\textit{L-INF}}$ *let* $A_\varphi(i, w_I) = \{v \in R_{c,i}(w_I) \mid \varphi \in v\}$*. Then, we put* $N_c(i, w_I) = \{A_\varphi(i, w_I) \mid \mathbf{B}_i\, \varphi \in w_I\}$*;*
- $E_c : Agt \times W_c \longrightarrow 2^{\mathcal{L}_{\mathsf{ACT}}}$ *is such that, for each* $i \in Agt$ *and* $w_I, v_I \in W_c$*, if* $w_I R_{c,i} v_I$ *then* $E_c(i, w_I) = E_c(i, v_I)$*;*
- $B_c : Agt \times W_c \longrightarrow \mathbb{N}$ *is such that, for each* $i \in Agt$ *and* $w_I, v_I \in W_c$*, if* $w_I R_{c,i} v_I$ *then* $B_c(i, w_I) = B_c(i, v_I)$*;*
- $C_c : Agt \times \mathcal{L}_{\mathsf{ACT}} \times W_c \longrightarrow \mathbb{N}$ *is such that, for each* $i \in Agt$*,* $\alpha \in \mathcal{L}_{\mathsf{ACT}}$*, and* $w_I, v_I \in W_c$*, if* $w_I R_{c,i} v_I$ *then* $C_c(i, \alpha, w_I) = C_c(i, \alpha, v_I)$*;*
- $A_c : Agt \times W_c \longrightarrow 2^{Atm_A}$ *is such that, for each* $i \in Agt$ *and* $w_I, v_I \in W_c$*, if* $w_I R_{c,i} v_I$ *then* $A_c(i, w_I) = A_c(i, v_I)$*;*
- $H_c : Agt \times W_c \longrightarrow 2^{Atm_A}$ *is such that, for each* $i \in Agt$ *and* $w_I, v_I \in W_c$*, if* $w_I R_{c,i} v_I$ *then* $H_c(i, w_I) = H_c(i, v_I)$*;*
- $P_c : Agt \times W_c \times Atm_A \longrightarrow \mathbb{N}$ *is such that, for each* $i \in Agt$ *and* $w_I, v_I \in W$*, if* $w_I R_{c,i} v_I$ *then* $P_c(i, w_I, \phi_A) = P_c(i, v_I, \phi_A)$*;*
- $V_c : W_c \longrightarrow 2^{Atm}$ *is such that* $V_c(w_I) = Atm \cap w_I$*;*
- T_c *: the time function defined as before.*

Analogously to what done before, let $R_{c,i}(w_I)$ denote the set $\{v_I \in W_c \mid w_I R_{c,i} v_I\}$, for each $i \in Agt$. M_c is an *L-INF* model as defined in Definition 2.1, since, it satisfies conditions

(**C1**),(**C2**),(**D1**),(**E1**),(**F1**),(**G1**),(**G2**),(**H1**). Hence, it models the axioms and the inference rules 1–19 and 23 introduced before (while, as mentioned in Section 2.5, axioms 20–22 are assumed to be enforced by the specification of agents behaviour). Consequently, the following properties hold too. Let $w_I \in W_c$, then:

- given $\varphi \in \mathcal{L}_{\textit{L-INF}}$, it holds that $\mathbf{K}_i\,\varphi \in w_I$ if and only if $\forall v_I \in W_c$ such that $w_I R_{c,i} v_I$ we have $\varphi \in v$;
- for $\varphi \in \mathcal{L}_{\textit{L-INF}}$, if $\mathbf{B}_i\,\varphi \in w_I$ and $w_I R_{c,i} v$ then $\mathbf{B}_i\,\varphi \in v_I$;

Thus, $R_{c,i}$-related worlds have the same knowledge and N_c-related worlds have the same beliefs, i.e. there can be $R_{c,i}$-related worlds with different beliefs.

By proceeding similarly to what done in [16], we obtain the proof of strong completeness. For lack of space, we list the main theorems but omit lemmas and proofs, that we have however developed analogously to what done in previous work [10].

Theorem 3.1. *L-INF is strongly complete for the class of L-INF models.*

Theorem 3.2. *L-DINF is strongly complete for the class of L-INF models.*

4. Conclusions

In this paper we proposed a possible way to enrich the epistemic logic introduced in [9, 10, 11], originally designed to express group dynamics of cooperative agents, with the possibility of specifying time intervals to express the time periods in which agents' acting takes place. Hence, by adapting the treatment introduced in [6], an enriched semantics for formulas, as well as a new belief update mechanism, has been suitably designed for the new temporalized logic. The approach appears promising and its usefulness has been shown by outlining a not trivial example.

References

[1] J. Rocha (Ed.), Multi-agent Systems, IntechOpen, 2017. doi:`10.5772/66595`.

[2] M. Fisher, R. H. Bordini, B. Hirsch, P. Torroni, Computational logics and agents: A road map of current technologies and future trends, Comput. Intell. 23 (2007) 61–91. doi:`10.1111/j.1467-8640.2007.00295.x`.

[3] R. Calegari, G. Ciatto, V. Mascardi, A. Omicini, Logic-based technologies for multi-agent systems: a systematic literature review, Auton. Agents Multi Agent Syst. 35 (2021) 1. doi:`10.1007/s10458-020-09478-3`.

[4] S. Costantini, A. Formisano, V. Pitoni, Timed memory in resource-bounded agents, in: C. Ghidini, B. Magnini, A. Passerini, P. Traverso (Eds.), Proc. of AI*IA 2018, volume 11298 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 15–29.

[5] S. Costantini, V. Pitoni, Memory management in resource-bounded agents, in: M. Alviano, G. Greco, F. Scarcello (Eds.), Proc. of AI*IA 2019, volume 11946 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 46–58.

[6] V. Pitoni, S. Costantini, A temporal module for logical frameworks, in: B. Bogaerts, E. Erdem, P. Fodor, A. Formisano, G. Ianni, D. Inclezan, G. Vidal, A. Villanueva, M. De Vos, F. Yang (Eds.), Proc. of ICLP 2019 (Tech. Comm.), volume 306 of *EPTCS*, 2019, pp. 340–346.

[7] D. G. Pearson, R. H. Logie, Effects of stimulus modality and working memory load on mental synthesis performance, Imagination, Cognition and Personality 23 (2003) 183–191.

[8] R. H. Logie, Visuo-Spatial Working Memory, Psychology Press, Essays in Cognitive Psychology, 1994.

[9] S. Costantini, V. Pitoni, Towards a logic of "inferable" for self-aware transparent logical agents, in: C. Musto, D. Magazzeni, S. Ruggieri, G. Semeraro (Eds.), Proceedings of the Italian Workshop on Explainable Artificial Intelligence co-located with 19th International Conference of the Italian Association for Artificial Intelligence, 2020, volume 2742 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 68–79.

[10] S. Costantini, A. Formisano, V. Pitoni, An epistemic logic for multi-agent systems with budget and costs, in: W. Faber, G. Friedrich, M. Gebser, M. Morak (Eds.), Logics in Artificial Intelligence - 17th European Conference, JELIA 2021, Proceedings, volume 12678 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 101–115. doi:10.1007/978-3-030-75775-5_8.

[11] S. Costantini, A. Formisano, V. Pitoni, An epistemic logic for modular development of multi-agent systems, in: N. Alechina, M. Baldoni, B. Logan (Eds.), Engineering Multi-Agent Systems 9th International Workshop, EMAS 2021, Revised Selected papers, Lecture Notes in Computer Science, Springer, 2022.

[12] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Systems 2 (1990) 255–299.

[13] A. S. Rao, M. Georgeff, Modeling rational agents within a BDI architecture, in: Proc. of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann, 1991, pp. 473–484.

[14] H. V. Ditmarsch, J. Y. Halpern, W. V. D. Hoek, B. Kooi, Handbook of Epistemic Logic, College Publications, 2015. Editors.

[15] R. W. Weyhrauch, Prolegomena to a theory of mechanized formal reasoning, Artificial Intelligence 13 (1980) 133–170.

[16] P. Balbiani, D. F. Duque, E. Lorini, A logical theory of belief dynamics for resource-bounded agents, in: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, AAMAS 2016, ACM, 2016, pp. 644–652.

An Application of ASP for Procedural Content Generation in Video Games

Andrea De Seta, Mario Alviano

DEMACS, University of Calabria, Via Bucci 30/B, 87036 Rende (CS), Italy

Abstract

Procedural content generation eases and accelerates the development of video games by creating data algorithmically through a combination of human-generated assets and algorithms usually coupled with computer-generated randomness. This paper presents a use case of Answer Set Programming (ASP) for procedural content generation of levels in a rougelike video game powered by the Godot Engine. The main elements of a set of human-generated rooms are represented by ASP facts, among them positions of doors, presence of treasures and power-ups. Within this knowledge, ASP is asked to generate dungeons satisfying a few conditions, among them the correct positioning of rooms, the absence of unreachable rooms and constraints on the occurrences of rooms. Scalability of ASP in this context is evaluated empirically, showing that it can generate in few seconds levels that comprise thousands of rooms.

1. Introduction

Video game industry, i.e. the industry involved in the development, marketing and monetization of video games, has grown sensibly in the recent years, and its annually generated sales are in the order of hundreds of billions of dollars. The development of a video game is the first step to enter such an industry, and several frameworks are nowadays available to start with a set of common primitives that can be combined to achieve appealing results in relatively short time. Among them there is the Godot Engine (https://godotengine.org/), a completely free and open-source game engine under MIT license, providing a huge set of common tools that, especially for 2D-games, significantly accelerate all the development phase.

Further acceleration in the development of video games can be provided by techniques that go under the name of *procedural content generation (PCG)*, and essentially consist of algorithms that automatically create levels, maps, weapons, background scenery, and music for video games [1, 2]. The idea is not new, and actually exploited already in the '70s in historical titles such as PEDIT5 (https://en.wikipedia.org/wiki/Pedit5) to circumvent the limited amount of computational resources [3]. Many video games followed the idea of PEDIT5 and took advantage of PCG. Later, those video games were classified as *rougelike*, a term originated from '90s USENET newsgroups. Even if the exact definition of a rougelike game remains a point of debate in the video game community, some characteristics are clearly identified. Specifically, rougelike is a subgenre of role-playing video games characterized by a dungeon crawl through PCG levels, turn-based gameplay, grid-based movement, and permanent death of the player character.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

In this work the term rougelike is slightly abused and used to refer to video games that have all the aforementioned characteristics but turn-based gameplay and grid-based movement. Stated differently, in the following we consider a video game characterized by a dungeon crawl through PCG levels, *real-time gameplay*, *grid-based positioning but free movements*, and permanent death of the player character. The presented game is entitled WizardSet, is powered by the Godot Engine and takes advantage of Answer Set Programming (ASP; [4, 5, 6]) for PCG of levels. In a nutshell, the idea underlying WizardSet is to design some elements of the video game with the Godot Engine, represent such elements in a format understandable by ASP systems, and then combine such a representation with an ASP encoding to delegate the generation of a complete level to an ASP reasoner. This way, levels are generated when needed, providing a unique experience to the player for each game. After introducing the required background knowledge (Section 2), we will detail on the PCG implemented in WizardSet (Section 3), and report on the result of an experiment aimed at assessing the scalability of the proposed procedure (Section 4).

2. Background

The Godot Engine provides an Integrated Development Environment to design scenes, where a scene represents an element of the video game and is characterized by a tree and possibly a script. The tree of a scene is made of nodes of different nature that can be used to associate sprites and physical properties to the element of the video game. Scripts are usually written in GDScript, a high-level, dynamically typed programming language with a syntax similar to Python (but a few other languages are supported as well).

ASP is a rule-based language supporting object variables and negation under stable model semantics. Object variables are removed by means of intelligent grounding, and stable models are searched by conflict-driven clause learning algorithms. ASP systems extend the basic language of ASP with several constructs for representing common knowledge, among them integer arithmetic and aggregates. The reader is referred to the ASP Core 2 [7] format for details.

3. Procedural Content Generation within ASP

WizardSet represents a few key features of its rooms in terms of ASP facts:

- `room(`r`)`, for each available room, where r is a positive integer identifying the room (the current release of WizardSet provides 17 rooms with different layout and features);
- `room_door(`r`,`l`)`, if room r has a door in the wall l, where l is one of `north`, `south`, `west`, and `east`;
- `room_flag(`r`,`f`)`, if room r has feature f, where f is among `initial`, `boss`, and `treasure`;
- `flag_bounds(`f`,`min`,`max`)`, for each represented feature, where min and max are integers to bound the number of rooms with feature f in a level; in particular, we fix `flag_bounds(initial,1,1)`, `flag_bounds(boss,1,1)`, and `flag_bounds(treasure,0,1)`, i.e. there must be exactly one initial room and boss room, and at most one treasure room in each generated level.

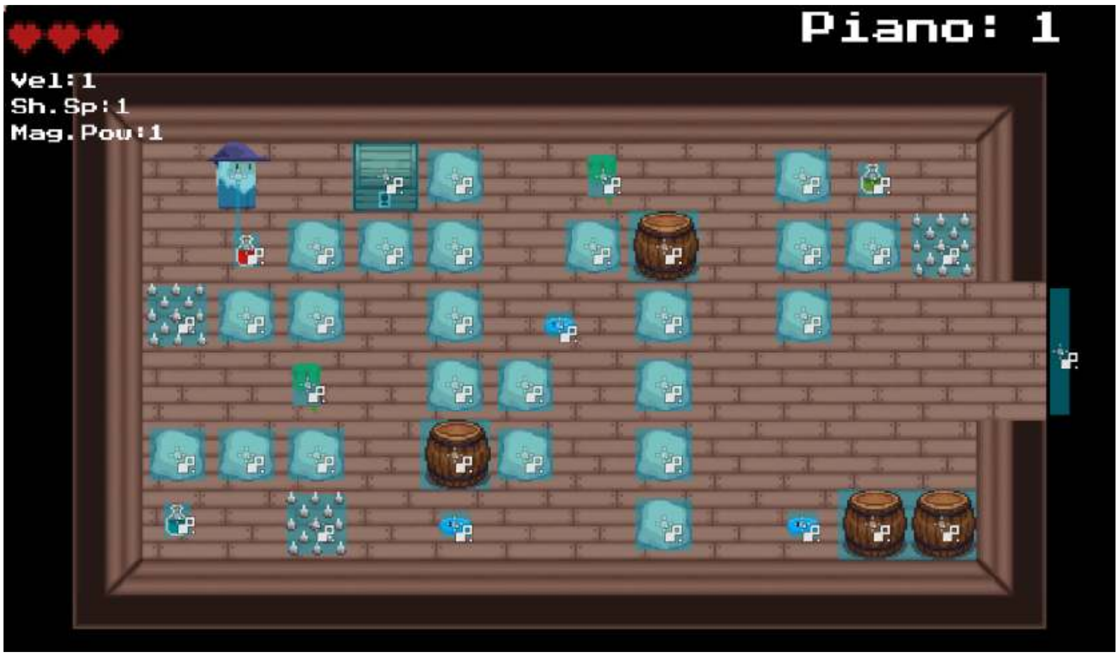


Figure 1: Design view of room 4 of WizardSet, having a door in the east wall and containing a treasure. The room is represented by the following ASP facts: `room(4)`, `room_door(4,east)`, and `room_flag(4,treasure)`.

An example room and the associated ASP facts are reported in Figure 1. Additionally, the ASP encoding is enriched with the following facts to represent movements in the four cardinal directions:

```
delta(north,-1, 0).     opposite(north,south).
delta(south, 1, 0).     opposite(south,north).
delta(west,  0,-1).     opposite(west, east).
delta(east,  0, 1).     opposite(east, west).
```

The reasoning core of the ASP encoding empowering WizardSet is parameterized by the size of the grid to generate (constants `rows` and `cols`), the number of rooms to deploy (constant `required_rooms`), and the health points of the player (constant `hp`). The ASP encoding non-deterministically assigns rooms to cells of the grid, ensuring that a few constraints are satisfied. The following ASP rules are used (and described below):

```
r1 :  {assign(X,Y,nil); assign(X,Y,R) : room(R)} = 1 :-
            X = 1..rows, Y = 1..cols.
r2 :  assign(0,     Y,     nil) :- Y = 0..cols+1.
r3 :  assign(rows+1,Y,     nil) :- Y = 0..cols+1.
r4 :  assign(X,     0,     nil) :- X = 0..rows+1.
r5 :  assign(X,     cols+1,nil) :- X = 0..rows+1.
r6 :  :- #count{X,Y : assign(X,Y,R), R != nil} != required_rooms.
r7 :  :- assign(X,Y,R), room_door(R,L), delta(L,DX,DY),
```

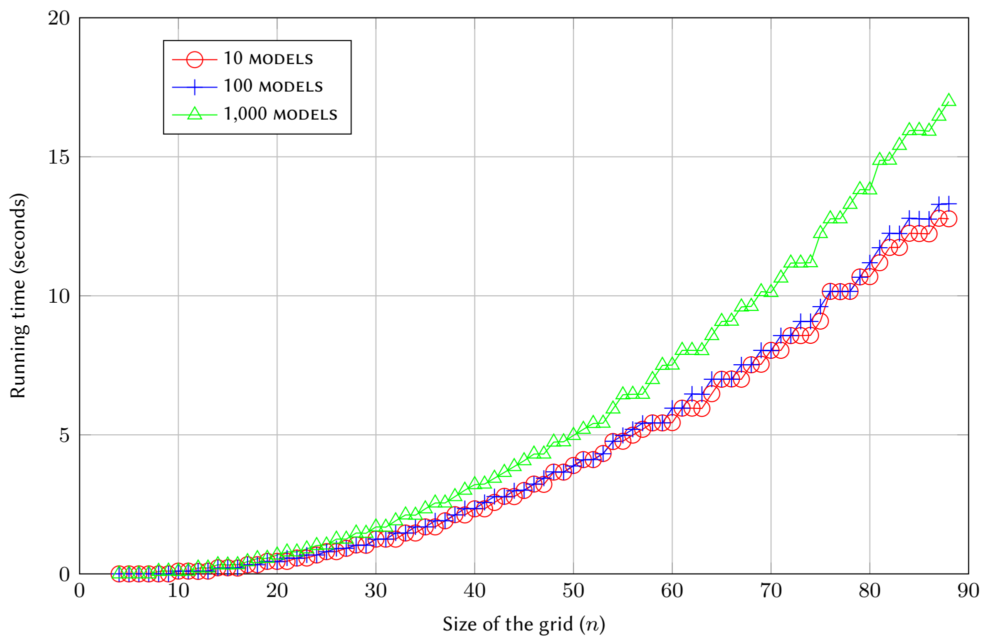


Figure 2: CLINGO performance on $n \times n$ level generation

```
           assign(X+DX,Y+DY,R'), opposite(L,L'), not room_door(R',L').
r8  :   :- flag_bounds(F,MIN,MAX),
           not MIN <= #count{X,Y : assign(X,Y,R), room_flag(R,F)} <= MAX.
r9  :   reachable(X,Y) :- room_flag(R,initial), assign(X,Y,R).
r10 :   reachable(X+DX,Y+DY) :- reachable(X,Y), assign(X,Y,R),
                                room_door(R,L), delta(L,DX,DY).
r11 :   :- assign(X,Y,R), R != nil, not reachable(X,Y).
r12 :   spawn_hp_potion(R) :- room_flag(R,initial), hp <= 2.
```

The assignment itself is generated by rule r_1, which associates each cell with a room or with the `nil` value (to denote a non-playable cell of the grid, i.e. a cell not containing any room). Rules r_2–r_5 introduce a border of non-playable cells, so to clearly delimit the playable boundary of the generated level. Rule r_6 enforces that the number of assigned rooms is the required one, and rule r_7 ensures the correct placement of doors. Rule r_8 ensures that the presence of represented features in the required amount. Rules r_9–r_{11} impose that all playable cells are actually connected. Finally, rule r_{12} provides an example of power-up that can be placed in some rooms of the generated dungeon; in this case, the power-up is a health potion to be placed in the initial room if the player starts with few health points.

4. Implementation and Experiment

WizardSet is open-source (https://github.com/Tatanka4/WizardSet). The ASP system adopted for PCG is CLINGO 5.5.0 [8], and communication between the Godot

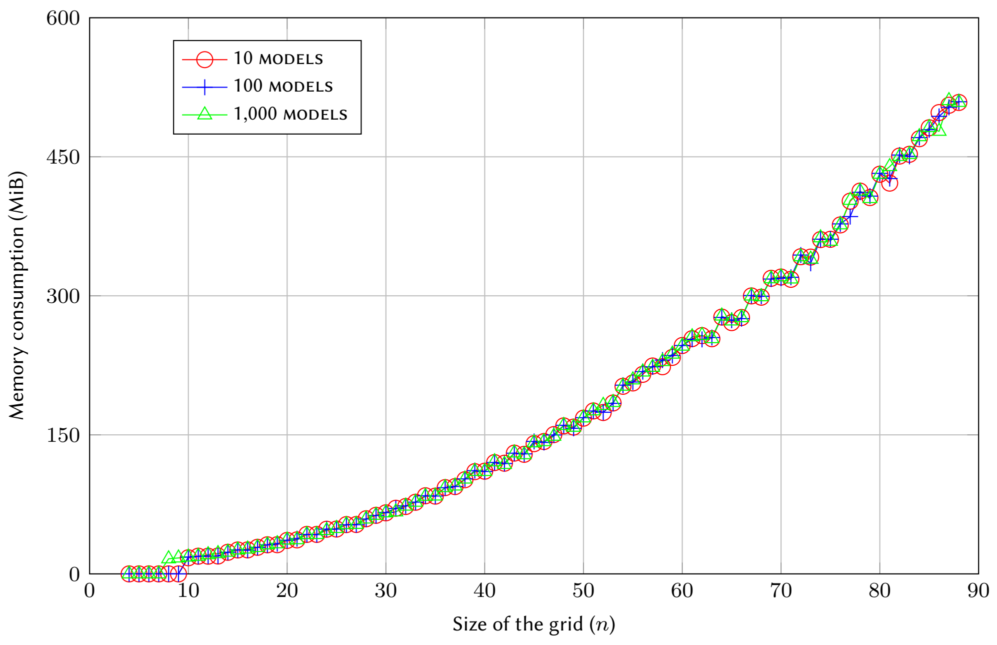


Figure 3: CLINGO memory consumption on $n \times n$ level generation

Engine and CLINGO is achieved by synchronous system calls and file sharing. Parameters for the generation of levels are adjusted to increase the size of the grid and the number of playable cells, and therefore the expected difficulty. On ordinary playing sessions the time required for PCG is negligible, and usually below a second. Therefore, in order to empirically evaluate the scalability of the proposed ASP encoding, we designed a first experiment to measure the execution time required by CLINGO to generate squared grids of increasing size $n \times n$, with the number of requested rooms set to $\frac{n \times n}{2}$. All tests are run on an Intel Xeon 2.4 GHz with 16 GiB of memory. Time and memory were limited to 20 seconds and 4 GiB (WIZARDSET is expected to be run on laptops and low-end PCs, and level generation must stay in the order of a few seconds).

Results are shown in Figure 2 for different settings of the number of models asked to CLINGO; in fact, to increase randomness in the generation of levels, WIZARDSET selects one model among several that are produced by CLINGO. Additionally, we observed that the generation of levels is unfeasible without limiting the number of models; the reason is to be attributed to the high number of possible models, which is already 2,064 for grids of size 4×4 (0.1 seconds of computation) and 1,360,822 for grids of size 5×5 (89 seconds of computation). We therefore ran all test cases by limiting the number of models to 10, 100 and 1,000. Within these limits, levels can be generated in less than 20 seconds up to grids of size 89×89. Figure 3 reports the memory consumption for producing 10, 100 and 1,000 models, and it can be observed that almost the same consumption was measured. Given the results of our experiment, in the release of WIZARDSET we fixed the number of models to ask to CLINGO to 1,000 models, as there is no significant performance gain in asking for less models.

5. Conclusion

Previous works in the literature have already shown applications of ASP to video games. For example, ASP is used in Angry-HEX [9], an AI that can play Angry Birds, and in ThinkEngine [10, 11], an integration of ASP in Unity. ASP can be used profitably for PCG in video games as well, and the idea was already used in the literature [12, 13, 14]. This work provides another concrete example of ASP-powered PCG by presenting the first combination of the Godot Engine with an ASP system to generate levels of a rougelike video game. In our experience, the main advantage of adopting ASP for this task relies in the declarative power of the language, thanks to which constraints and desiderata for PCG can be specified succinctly in a few lines of code. A similar observation is reported in [14], whose approach to generate dungeons is to partition the space in rectangular areas, and then generate a random room in each area; in this case, the ASP encoding models desiderata on the generation of rooms. In our approach, rooms are defined by the programmer, their features represented in ASP, and such a knowledge base is used to generate a dungeon. Moreover, our ASP representation is suitable for future extensions of the approach, as for example by enriching the knowledge base with other features of the rooms so to control the number of enemies and power-ups in the generated levels.

Acknowledgments

This work was partially supported by the projects PON-MISE MAP4ID (CUP B21B19000650008) and PON-MISE S2BDW (CUP B28I17000250008), by the LAIA lab (part of the SILA labs) and by GNCS-INdAM.

References

[1] M. Hendrikx, S. A. Meijer, J. V. D. Velden, A. Iosup, Procedural content generation for games: A survey, ACM Trans. Multim. Comput. Commun. Appl. 9 (2013) 1:1–1:22. doi:10.1145/2422956.2422957.

[2] J. Togelius, G. N. Yannakakis, K. O. Stanley, C. Browne, Search-based procedural content generation: A taxonomy and survey, IEEE Trans. Comput. Intell. AI Games 3 (2011) 172–186. doi:10.1109/TCIAIG.2011.2148116.

[3] N. Brewer, Computerized dungeons and randomly generated worlds: From rogue to minecraft, Proc. IEEE 105 (2017) 970–977. doi:10.1109/JPROC.2017.2684358.

[4] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (1991) 365–386. doi:10.1007/BF03037169.

[5] V. W. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm – A 25-Year Perspective, Springer Verlag, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.

[6] I. Niemelä, Logic programming with stable model semantics as constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (1999) 241–273. doi:10.1023/A:1018930122475.

[7] F. Calimeri, et al, Asp-core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309. doi:10.1017/S1471068419000450.

[8] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: M. Carro, A. King (Eds.), TC of ICLP'16, volume 52 of *OASIcs*, Schloss Dagstuhl, Germany, 2016, pp. 2:1–2:15.

[9] F. Calimeri, et al., Angry-hex: An artificial player for angry birds based on declarative knowledge bases, IEEE Trans. Comput. Intell. AI Games 8 (2016) 128–139. doi:10.1109/TCIAIG.2015.2509600.

[10] D. Angilica, G. Ianni, F. Pacenza, Tight integration of rule-based tools in game development, in: M. Alviano, G. Greco, F. Scarcello (Eds.), AI*IA 2019, Rende, Italy, November 19-22, 2019, Proceedings, volume 11946 of *LNCS*, Springer, 2019, pp. 3–17. doi:10.1007/978-3-030-35166-3_1.

[11] F. Calimeri, S. Germano, G. Ianni, F. Pacenza, S. Perri, J. Zangari, Integrating rule-based AI tools into mainstream game development, in: C. Benzmüller, F. Ricca, X. Parent, D. Roman (Eds.), RuleML+RR 2018, Luxembourg, September 18-21, 2018, volume 11092 of *LNCS*, Springer, 2018, pp. 310–317. doi:10.1007/978-3-319-99906-7_23.

[12] X. Neufeld, S. Mostaghim, D. Perez Liebana, Procedural level generation with answer set programming for general video game playing, 2015, pp. 207–212. doi:10.1109/CEEC.2015.7332726.

[13] A. M. Smith, M. Mateas, Answer set programming for procedural content generation: A design space approach, IEEE Trans. Comput. Intell. AI Games 3 (2011) 187–200. doi:10.1109/TCIAIG.2011.2158545.

[14] F. Calimeri, S. Germano, G. Ianni, F. Pacenza, A. Pezzimenti, A. Tucci, Answer set programming for declarative content specification: A scalable partitioning-based approach, in: AI*IA, volume 11298 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 225–237.

An Intelligent Ecosystem to improve Patient Monitoring using Wearables and Artificial Intelligence[⋆]

Stefania Costantini[1,3,*], Fabio Persia[1] and Lorenzo De Lauretis[1]

[1]*DISIM - Università dell'Aquila, via Vetoio–loc. Coppito, 67100 L'Aquila, Italy*
[3]*GNCS-INdAM, piazzale Aldo Moro 5, 00185 Roma, Italy*

Abstract
Our work describes a smart-ecosystem able to monitor patients' health condition, even at home or at work, by exploiting a creative blend of Medical Wearables, Intelligent Agents, Complex Event Processing and Image Processing. With the help of a smart application, that links together the Wearables and the power of Artificial Intelligence, patients will be continuously and actively supervised during their daily activities. This can even save their lives, in case sudden or gradual issues should occur. Thanks to our system, patients with non-severe though potentially unstable chronic diseases will no longer overburden first aid services. This is also useful for containing the spread of COVID-19. Specifically, in this paper we focus on automated vitals monitoring, electrocardiogram (ECG) analysis, and Psoriasis detection.

Keywords
Artificial Intelligence, Wearables, Intelligent Agents, Complex Event Processing, Image Processing

1. Introduction

The Coronavirus pandemic has highlighted telehealth as a crucial component of modern and sustainable treatment. In fact, telemedicine's primary purpose is to virtually erase the distance between patient and physician, as well as to reduce time and expense involved in healthcare access. Furthermore, during the COVID-19 pandemic, the growing usage of telehealth has often reduced the danger of providers and patients being exposed to the virus [1]. However, despite recent efforts, telemedicine is still in its early stages for a variety of reasons; as a result, one of the most significant consequences is that First Aid departments are frequently overburdened by people who do not require immediate assistance; this could be avoided by allowing patients to use telehealth applications to monitor their vitals either at home or even at work. Thus, in order to cope with these issues, in our approach we joined together Wearables (such as portable ECG devices and Pulsoximeters), Intelligent Agents, Complex Event Processing (CEP), and Image Processing algorithms in an integrated framework able to follow the patient wherever (s)he

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
[⋆]Research partially supported by project INDAM GNCS-2022 InSANE (CUP_E55F22000270001).
[*]Corresponding author.
stefania.costantini@univaq.it (S. Costantini); fabio.persia@univaq.it (F. Persia);
lorenzo.delauretis@graduate.univaq.it (L. D. Lauretis)
0000-0002-5686-6124 (S. Costantini)

is. This paper improves our previous work in the E-Health field [2, 3], by adding new features, algorithms and devices, in order to create a complete ecosystem.

Specifically, our ecosystem is composed by a number of devices (both hardware and software) allowing users to monitor their health status constantly. We have sensors measuring oxygen saturation, hearth rate, and all the heart parameters (by performing an ECG); sensors communicate with the developed Android application, that forwards the acquired data to a server. Such data will be immediately analyzed by the main components of our system, that, through sophisticated algorithms, are able to assess the patient's health status. The assessment is provided to him/her, and, in case of severe detected problems, a doctor is immediately informed, and, if necessary, first aid services are alerted.

In the literature, in [4] and [5] specific systems are described that can be used to detect Atrial Fibrillation and hearth anomalies in patients, via wearable devices, such as an Arduino linked to a Bitalino Board or smart sensors embedded in dresses.

In our work, instead, we exploit medical-grade more precise ECG sensors, allowing the user to autonomously perform a professional ECG with very accurate outcomes; additionally, these are ready-to-use devices that the user does not need to assemble.

In [6], the authors describe a Multi-Agent system for E-Health that can be used to read and analyze sensor values, and alert an administrator if health issues are detected. Our system improves their work by adding wearable technology, thus providing the patient with the possibility to monitor his/her health status in every situation, not only at home.

Moreover, in [7] the authors describe a method that can be adopted in order to detect the Psoriasis on the patient's skin, using k-means clustering, segmentation and bounding box strategies. In our work, differently from them, we detected Psoriasis using AVG color detection, image segmentation and clustering, obtaining, in addition, even the severity value of the psoriasis detected on the hand.

In the majority of these works, even if they may appear similar to ours, they implement stand-alone solutions with a single “brain", monitoring a limited number of situations; in our work, instead, we created a smart-ecosystem, with more “brain-sectors" linked together, each one specifically needed for some kind of calculations, that, together, have a very strong computational power.

In summary, our contributions are listed in the following:

- We present a wearable smart-ecosystem, that can follow users wherever they go, even to their job;
- We integrate Intelligent Agents, Complex Event Processing and Wearables in a joint mobile application, leaving the computational power to the Server-side;
- We define a specific Image Processing algorithm, able to detect whether the user has Psoriasis or not, which will be also extended to other pathologies in the near future.

2. System Architecture

Our system consists of six main components: User and Doctor Interfaces - Android App, Database and Web interface, Hardware, Intelligent agents, Complex Event Processing and Image Processing.

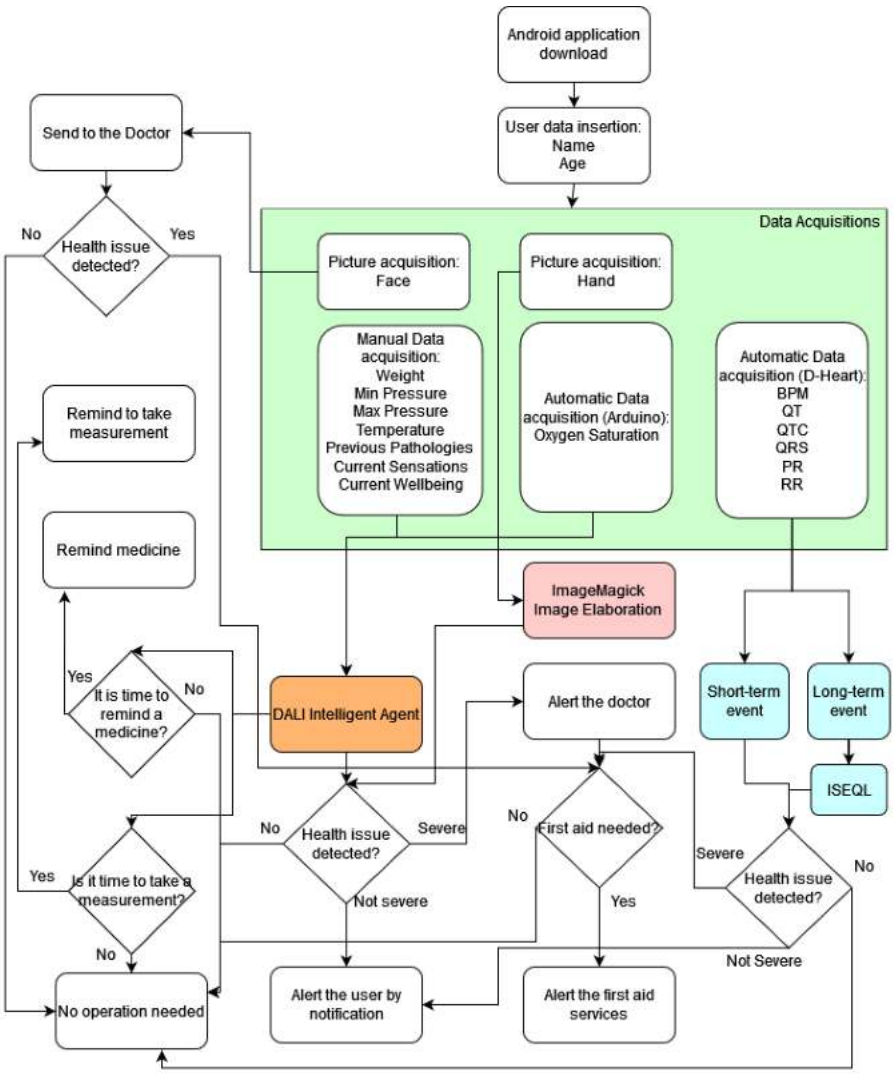

Android application download
User data insertion: Name Age
Data Acquisitions
Send to the Doctor
Health issue detected?
No
Yes
Picture acquisition: Face
Picture acquisition: Hand
Manual Data acquisition: Weight Min Pressure Max Pressure Temperature Previous Pathologies Current Sensations Current Wellbeing
Automatic Data acquisition (Arduino): Oxygen Saturation
Automatic Data acquisition (D-Heart): BPM QT QTC QRS PR RR
Remind to take measurement
Remind medicine
ImageMagick Image Elaboration
It is time to remind a medicine?
DALI Intelligent Agent
Alert the doctor
Short-term event
Long-term event
ISEQL
First aid needed?
Severe
Not severe
Is it time to take a measurement?
Not Severe
No operation needed
Alert the user by notification
Alert the first aid services

Figure 1: Whole system workflow

The user and doctor interfaces consist in a mobile application, which is currently released only for android devices, but that we are planning to customize for Apple devices too. Our system's "core" is the database, that, in our case, is a MySQL one. A web interface that runs the main algorithm which acts as an effective "hub" linking together the Android Applications, Intelligent agents, CEP, Image processing and the Database.

Our ecosystem encompasses a number of hardware devices (medical wearables), with the advantage of being modular. In fact, being also equipped an Arduino, we can attach a number of sensors that are very effective for our goals; we have already used Arduino with other medical-related and wearable-related projects [8, 9], obtaining interesting results that improved our awareness to use wearables in the medical field. We currently have one sensor attached to

Arduino (the pulse oximeter), and one sensor that is a single block (the ECG device), but we are going to expand this collection in the near future. They both communicate to the mobile app via Bluetooth.

Additionally, after an accurate search for an appropriate ECG device, supported by qualified medical advice, we selected *D-Heart*[1]. Specifically, D-Heart is the first ECG device for smartphone that is simple to use, clinically reliable, portable and affordable. To sum up, Fig. 1 depicts the entire workflow of our system.

2.1. Intelligent Agents

In our work, we use DALI in order to implement a system composed by intelligent agents. DALI is an agent-oriented logical language derived from Prolog, and, similarly to it, is based on the logic programming paradigm. DALI has been fully implemented, on the basis of a fully logical semantics [10]. By analyzing data, our intelligent system is able to recognize the patient's state of well-being or discomfort in both short and long term; therefore, the system is able to assess the seriousness of the situation and, if necessary, alert a human doctor or even call the Emergency Service. Using DALI, we created two intelligent agents: the *Patient_agent* and the *Doctor_agent*. The Patient_agent analyses the patient's vital parameters (heart rate, saturation, minimum and maximum pressure, temperature, weight) taken by wearable devices or by the patient, and returns immediate feedback on each of them.

It also acts as an Ehealth-companion, reminding the user to take pills at the correct time, sending a reminder for temperature, blood pressure and weight measurements. The Doctor_agent receives messages from the Patient_agent, analyzes the communicated problem, e.g., fever or tachycardia, and responds by suggesting a medication to take or, in case of serious danger, it suggests going to the Emergency Room. Events currently detected by our DALI intelligent agents are *Angina Pectoris*, *Tachicardia*, *Bradhicardia*, *Hypoxia*, *Hypertension* and *Fever*.

2.2. Complex Event Processing

The specific language that we use in order to carry out Complex Event Processing tasks extends relational algebra [11] and the well-known Allen's interval relationships [12]; specifically it is named ISEQL [13], standing for Interval-based Surveillance Event Query Language. We exploit this language so as to easily define the medical events we are interested in. Using ISEQL, we are able to detect short-term and long-term events in patient's ECGs.

2.2.1. Short-term event detection

A *Short-term Event* is an event to be detected within the context of a single ECG measurement. To detect such events, we must analyze the ECG signal in depth using specific criteria stored in the knowledge base and complying with the guidelines found in the literature [14]. As a result, the events automatically discovered in this stage thanks to the *D-Heart* device[2] are the *QT interval*, the *QTC interval*, the *RR interval*, the *QRS interval* and the *PR interval*.

[1]https://www.d-heartcare.com/it/
[2]https://www.d-heartcare.com

Moreover, on top of such events, we define specific additional event models in ISEQL, aimed at identifying possible anomalies in ECGs, thus extending the literature in the context of their automatic analysis. More specifically, the events are modeled in ISEQL and associated with their severities on a scale from 0 (lowest severity) to 3 (highest severity - in such a case, a medical doctor is automatically alerted).

2.2.2. Long-term event detection

Working on aggregated data at a different resolution, such as examining the results of several ECG measurements taken in specified wider temporal windows, allows a detailed long-term analysis of the patient's clinical history. In this paper, we focus on the detection of different categories of long-term events; in fact, on the one hand we identify some possible long-term anomalies exploiting ISEQL, and on the other hand we implement specific algorithms for detecting the *Atrial Fibrillation (AF)* and the *Wolff-Parkinson-White (WPW)* syndromes.

As regards the *Atrial Fibrillation* detection, we use the data that have been previously stored in our ECG measurements, such as RR intervals and heart-rate.

To sum up, we obtain the HR values and the standard deviation of the RR intervals in 5 ECGs, and, after applying our specific algorithm, we detect an AF disorder in case the obtained parameters are greater than a Threshold value[3]. The defined algorithm extends the one in [4], that encodes useful information on how to diagnose early Atrial Fibrillation (AF) using ECG measurements, with custom improvements and adaptations, via exploiting the full power of ISEQL on interval data.

As regards the Wolff-Parkinson-White syndrome detection, we exploit the information previously inferred from the ECG measurements, such as PR intervals and QRS complexes. For this kind of study, we have partially used and adjusted the algorithm obtained from [15], that contains useful information on how to diagnose Wolff-Parkinson-White syndrome (WPW) using ECG measurements. To sum up, we extract the PR intervals and the QRS complexes of 5 ECGs, then we apply an interval detection strategy on these events, and, if the conditions are met, we detect a WPW syndrome.

Additionally, exploiting the right cardinality constraint of ISEQL [16], the system is also able to alert patient and medical doctors in case anomalies are identified in several measurements in a relatively short temporal interval.

2.3. Image Processing

We also define an Image Processing algorithm in order to elaborate and extract features from images. We used image processing in order to evaluate the condition of the patient's hand, and possibly diagnose the presence of Psoriasis. In order to correctly elaborate the image of a person's hand, and make it "visible" to the computer, we exploit one of the most used software for Image elaboration, that is named ImageMagick[4].

[3]Both the number of measurements and threshold values were suggested by medical doctors; clearly, we can easily update them in case requirements will change.
[4]https://imagemagick.org

Our computer vision algorithm consists of three main steps. In the first one, we are going to detect the presence of the AVG colour of the Psoriasis on the hand of the patient, with a percentage of fuzzyness. In the second step, we fill the whole image with Black and White colours, through the technique of image segmentation, depending on the presence of the Psoriasis or not. In the third step, we cluster all the detected Psoriasis spots with a particular area threshold, using the technique of Clustering. After applying this algorithm, we obtain a certain number of psoriasis clusters detected; then, we are going to count their number and their size, and provide a response to the user according to the resulting data.

3. Experimental Results

After developing the prototype, we made a set of tests to validate its correctness and validity. So, we asked a doctor for assistance, asking for four patients with the following characteristics, that should be doing our tests:

1. The first patient should be a person who suffered from none of the below pathologies, nor other special or particular ones, being a “healthy" one. He should be a 45 years old male.
2. The second patient should be a person who suffered from Atrial Fibrillation, without other special or particular pathologies, in order to allow us to correctly detect his pathology, without being influenced by other factors. He should be a 45 years old male.
3. The third patient should be a person who suffered from Wolff-Parkinson-White syndrome, without other special or particular pathologies, in order to allow us to correctly detect his pathology, without being influenced by other factors. He should be a 45 years old male.
4. The fourth patient should be a female patient aged over 80, suffering from severe heart failures and occasional angina, with comorbidities.

We also asked the doctor for pictures of a hand with Psoriasis and a hand without Psoriasis, in order to test our image processing algorithm.

In order to get more precise results, we decided that a “blind" test would be more appropriate, in order to really understand the potentialities of our newborn system. So, we performed the first three tests like blind tests, the fourth one as a regular test, and the fifth one (the Psoriasis test) as a regular image test. In our blind tests, the operator of the device does not know the pathology of the person (s)he is examining, thus avoiding results contamination. In the regular test, the operator of the device knows the pathologies of the person (s)he is examining because those results are less “scientifically related" but more “development-related", so we decided to have a regular test instead. In the Psoriasis test, 10 hand pictures are given to the machine, some with psoriasis, some with no psoriasis, thus validating the capability of our machine to correctly discover the disease.

3.1. Healty Patient Case Study

In the first case, the patient that the operator examined was the healthy one, without any particular pathologies. He gave the patient the D-Heart device, the Arduino device, a pressure measurement device, a thermometer, and the Android smartphone used for testing.

In this test the patient did five ECGs, one every 45 minutes; the collected data related to all the ECGs were stored in the database. After running all the tests, the system was queried for results. The tests resulted as expected, being the patient the healthy one, and in fact the system did not discover any dangerous short-term or long-term events.

3.2. Atrial Fibrillation Case Study

In the second case, the patient that the operator examined was the one who suffers from Atrial Fibrillation, without any other particular pathologies. He gave the patient the same testing tools as the first patient. Also in this test the patient did five ECGs, one every 45 minutes.

Also in this case, the system confirmed that the patient suffers from Atrial Fibrillation, since it detected the related short-term and long-term events. The detected short-term events are listed in Table 1; as a result, there are a lot of “warnings", whose severity is 1, some dangerous situations, indicated with severity 2, and only one very severe situation, indicated with severity 3, with the id 233. Since an event of high severity is detected, immediately after the ECG data processing, the doctor is alerted. The long-term event detected, in this case, is the following: “An Atrial Fibrillation was discovered. The AVG BPM is 118.2 and the STD RR is 300.66". The system correctly detected all the short-term events, and, using ISEQL, it understood that the patient suffers from Atrial Fibrillation.

3.3. Wolff-Parkinson-White syndrome Case Study

In the third case study, the patient that the operator examined was the one who suffers from Wolff-Parkinson-White syndrome, without any other particular pathologies. He gave the patient the same testing tools as the other patients. Also within this test, the patient did five ECGs, one every 45 minutes.

In this case, the system, instead of detecting the WPW syndrome, discovered a “possible" WPW syndrome. The short-term events detected are listed in Table 2; as a result, there are a lot of “warnings", whose severity is 1, but no more severe situations detected; in this case, the doctor is not alerted, being the situation not “urgent". The long-term event detected, in this case, is the following: “A Potential Wolff-Parkinson-White syndrome was discovered. The AVG PR is 108.4, the AVG QRS is 125.8 and the STD QRS is 2.31". The system correctly detected all the short-term events, and, using ISEQL, it understood that the patient suffers from a possible Wolff-Parkinson-White syndrome, not fully understanding the problem, but going very close to detecting it.

3.4. Intelligent Agent Case Study

In the fourth case study, the patient that the operator examined was a female patient aged 85, suffering from severe heart failures and occasional angina, with comorbidities. In this case, differently from the previous ones, we are not going to do a precise diagnosis, but we are just testing the system, with particular interest to DALI and the related Intelligent Agent. In particular, we covered:

- The therapies (which medications at which time of the day).

Table 1
Atrial Fibrillation Case Study - short-term events detected

Ecg	ID	Name	Type	Value	Description	Severity
Ecg 1	212	Y	hr	105	low tachycardia	1
	213	Y	qt	435	slightly long	1
	214	Y	qtc	450	long	2
	215	Y	rr	850	regular	0
	216	Y	qrs	123	slightly long	1
	217	Y	pr	201	slightly long	1
Ecg 2	218	Y	hr	120	severe tachycardia	2
	219	Y	qt	350	regular	0
	220	Y	qtc	360	regular	0
	221	Y	rr	1450	slightly long	1
	222	Y	qrs	112	regular	0
	223	Y	pr	194	regular	0
Ecg 3	224	Y	hr	122	severe tachycardia	2
	225	Y	qt	340	regular	0
	226	Y	qtc	350	regular	0
	227	Y	rr	1150	regular	0
	228	Y	qrs	114	regular	0
	229	Y	pr	195	regular	0
Ecg 4	230	Y	hr	126	severe tachycardia	2
	231	Y	qt	346	regular	0
	232	Y	qtc	355	regular	0
	233	Y	rr	1650	too long	3
	234	Y	qrs	110	regular	0
	235	Y	pr	190	regular	0
Ecg 5	236	Y	hr	118	low tachycardia	1
	237	Y	qt	345	regular	0
	238	Y	qtc	356	regular	0
	239	Y	rr	950	regular	0
	240	Y	qrs	111	regular	0
	241	Y	pr	180	regular	0

- The symptoms that can be treated by readjusting the therapies.
- The situations of danger/alarm that require immediate intervention of a doctor, or urgent transportation to the hospital.

As a first step, the patient, aided by an assistant, since she is impractical with the use of smartphones, inserted all the data into the system, using the Android application.

The data the patients inserted, are listed in Table 3.

As a result, the patient has two parameters that can indicate health problems, i.e., the QT and the QTC intervals, which are slightly longer than normal ones. Then, the Intelligent Agent started doing its elaborations and reminding features. As first reasoning, the intelligent agent understood that the minimum pressure is too high, sending a notification to the patient's smartphone. After that, it continued with its reasonings, since it did not find anything else relevant.

At the beginning of the next day, the system began sending notifications to the patient's smartphone. The first notification, which arrived at 7, had the following text: "You should take Eutirox 50, quantity 1 capsule"; it means that the patient should take 1 pill of Eutirox 50, at 7:00. Later, it continued sending notifications to the patient, to remind her of the other

Table 2
Wolff-Parkinson-White syndrome Case Study - short-term events detected

Ecg	ID	Name	Type	Value	Description	Severity
Ecg 1	242	Z	hr	70	regular	0
	243	Z	qt	345	regular	0
	244	Z	qtc	355	regular	0
	245	Z	rr	650	regular	0
	246	Z	qrs	123	slightly long	1
	247	Z	pr	110	slightly short	1
Ecg 2	248	Z	hr	70	regular	0
	249	Z	qt	349	regular	0
	250	Z	qtc	350	regular	0
	251	Z	rr	654	regular	0
	252	Z	qrs	125	slightly long	1
	253	Z	pr	114	slightly short	1
Ecg 3	254	Z	hr	70	regular	0
	255	Z	qt	349	regular	0
	256	Z	qtc	350	regular	0
	257	Z	rr	654	regular	0
	258	Z	qrs	129	slightly long	1
	259	Z	pr	110	slightly short	1
Ecg 4	260	Z	hr	80	regular	0
	261	Z	qt	329	regular	0
	262	Z	qtc	350	regular	0
	263	Z	rr	640	regular	0
	264	Z	qrs	124	slightly long	1
	265	Z	pr	108	slightly short	1
Ecg 5	266	Z	hr	80	regular	0
	267	Z	qt	345	regular	0
	268	Z	qtc	355	regular	0
	269	Z	rr	620	regular	0
	270	Z	qrs	128	slightly long	1
	271	Z	pr	100	slightly short	1

medicines that she should take. At 14:00, the system reminded the patient that she should take the weight measurement, pressure measurement and temperature measurement. This time, the measurements were “perfect", and the system did not detect any event. At 17:45, the patient did not feel very well, and took the related measurements, registering a low minimum pressure, with a value of 55, and low maximum pressure, with a value of 89. At this point, the intelligent agent, following its rules, noticed that the patient suffers from Hypertension, and, when both min and max pressures are low, a dangerous situation may happen, so, after displaying a warning to the patient, the system also alerted the doctor. The doctor briefly read the alert on the application, and called the patient, telling her that an ECG is required. The patient, at this point, with the help of an assistant, attached D-Heart and did an ECG, sending the values to the system. At this point, the doctor read the ECG values, noting that were normal, with no particular issues, and called the patient to reassure him, because it was just a low-pressure situation. As we can detect from this last example, the intelligent agent acted as previsioned, helping the patient into the home monitoring, with minimum effort needed by both patient and doctor.

Table 3
Elderly patient case study

Parameter	Value
Name	W
Weight	67
Age	85
Temperature	37
Max Pressure	130
Min Pressure	91
Saturation	97
Pathology	Hypertension
Pathology	Angina
BPM	87
QT	435
QTC	445
QRS	112
PR	192
RR	720

3.5. Hand Psoriasis Case Study

In the fifth case study, differently from the others, we examined ten pictures of the back of the hands, given to us by the doctor. In 8 out of 10 cases, the pictures represented a hand with Psoriasis, the other 2 pictures represented a hand with no psoriasis. In these test cases, we are giving a diagnosis, thus detecting the hands in which psoriasis is present.

As a first step, we entered our Android application and began inserting the pictures of the hand given by the doctor, one per time, in order to let the system do its elaborations. Later, we exported and elaborated the database, in order to have a human-readable result, in the form of a table; in this table, we can see the picture identifier, the number of clusters detected, the diagnosis made by our system and the real diagnosis made from the doctor. The data are visible in Table 4.

As we can see from Table 4, our system correctly detected 8 out of 8 cases of psoriasis. In one of these, it detected possible psoriasis, since there were fewer clusters because the disease was in an initial state, and, in another case, it detected Severe Psoriasis, being the disease in a Severe situation. In the other 2 cases, the system correctly recognized that the hand is in a regular stage, detecting 0 and 1 clusters for such cases.

3.6. Precision and Recall

In order to calculate precision for our experiments, we used the well-known *Precision* and *Recall* formulas.

In the Healthy Patient test, reasoning with short-term events, we have 30 TruePositives results (considering as TruePositive all the “regular" values), and 0 FalsePositives (irregular values, because a healthy patient should not have those values far from normal ranges). Since we do not

Table 4
Psioriasis hand test

ID	Clusters	Machine Diagnosis	Severity	Real Diagnosis
1	29	Psoriasis	2	Psoriasis
2	45	Psoriasis	2	Psoriasis
3	33	Psoriasis	2	Psoriasis
4	19	Possible Psoriasis	1	Psoriasis
5	0	Regular	0	Regular
6	55	Severe Psoriasis	3	Psoriasis
7	39	Psoriasis	2	Psoriasis
8	1	Regular	0	Regular
9	23	Psoriasis	2	Psoriasis
10	43	Psoriasis	2	Psoriasis

have short-term events detected, we obtained a precision of 1, which is the maximum precision possible. We also have a recall of 1, having also 0 FalseNegatives, which is the maximum recall possible.

In the Atrial Fibrillation Patient test, reasoning with short-term events, following the algorithm for the atrial fibrillation discovery previously mentioned, we should consider as TruePositives results all the BPM values that are greater than 100. We cannot consider RR values as TruePositives, FalsePositives or FalseNegatives, because we use the Standard Deviation in order to detect AF. We should consider FalseNegatives all the BPM values that are smaller than 100. We do not have FalsePositives values at all. To sum up, also referring to Table 1, we have 5 TruePositives results, 0 FalseNegatives and 0 FalsePositives, thus obtaining a precision of 1, that is the maximum precision possible. We also get a recall of 1, having also 0 FalseNegatives.

In the Wolff-Parkinson-White Patient test, reasoning with short-term events, following the algorithm for the WPW discovery previously mentioned, we should consider as TruePositives results all the PR values that are smaller than 120 and all the QRS complexes that are greater than 120. We cannot consider QRS values as TruePositives, FalsePositives or FalseNegatives, because we use the Standard Deviation in order to detect WPW. We should consider FalseNegatives all the PR values that are greater than 120 and all the QRS complexes that are smaller than 120. We do not have FalsePositives values at all. To sum up, also referring to Table 2, we have 10 TruePositives results, 0 FalseNegatives and 0 FalsePositives, thus obtaining a precision of 1. We also get a recall of 1, having also 0 FalseNegatives.

In the Psoriasis test, reasoning with short-term events, following the algorithm for the Psoriasis discovery previously exposed, we should consider as TruePositives results all the events that have a machine-diagnosis of Psoriasis (Psoriasis, Severe Psoriasis) and the patient is currently affected by psoriasis. We should consider FalsePositives all the cases detected by the machine as Psoriasis (Psoriasis, Severe Psoriasis), but the patient has not psoriasis. We should consider FalseNegatives all the cases detected by the machine as Regular or Possible Psoriasis, but the patient does have Psoriasis. To sum up, also referring to Table 4, we have 7 TruePositives results, 1 FalseNegatives and 0 FalsePositives, thus obtaining a precision of 1. We

Table 5
All tests summary

	Healthy	Atrial	WPW	Psoriasis
True Positives	30	5	10	7
False Positives	0	0	0	0
False Negatives	0	0	0	1
Precision	1	1	1	1
Recall	1	1	1	0.875

have a recall of 0.875, having 1 FalseNegatives element, that is the row with ID 4 on the Table 4.

Eventually, In Table 5, we can see a brief summary of all our experiments, summarizing all the results of our tests, showing True Positives, False Positives, False Negatives, Precision and Recall for each test. All the tests obtained a precision of 1, indicating the maximum proportion of positive identifications is actually correct. Three out of four tests obtained a recall of 1, indicating that, for these three tests the maximum proportion of actual positives was identified correctly. For the other test, the Psoriasis detection one, we have a recall of 0.875, that, even being a high value, it is not the maximum value available, thus, we can improve our algorithm to obtain a higher recall value in the future.

4. Conclusion

In this paper, we have proposed an integrated framework aimed at actively supporting patient monitoring by exploiting an innovative combination of wearables, DALI intelligent agents and Complex Event Processing. Additionally, we also defined an effective image processing algorithm for psoriasis detection. Experiments conducted on real patients confirm the validity of the proposed approach.

The reader should notice that the health conditions and the case studies that we have discussed (all of them discussed with medical doctors) have been selected only to the aim to develop and test the system on solid ground. In the future, the system will be, again in concert with the doctors, extended in many ways. The intelligent agent will become able to cope with many other health issues; additional work will be also carried out to broaden the set of short-term and long-term events detectable via ISEQL. Addtionally, our Psoriasis detection algorithm will be able to automatically detect other skin diseases.

References

[1] D. M. Mann, J. Chen, R. Chunara, P. A. Testa, O. Nov, Covid-19 transforms health care through telemedicine: evidence from the field, Journal of the American Medical Informatics Association 27 (2020) 1132–1135.

[2] F. Persia, S. Costantini, C. Ferri, L. De Lauretis, D. D'Auria, A smart framework for

automatically analyzing electrocardiograms, in: 2021 Third International Conference on Transdisciplinary AI (TransAI), IEEE, 2021, pp. 64–67.

[3] S. Costantini, L. De Lauretis, C. Ferri, J. Giancola, F. Persia, A smart health assistant via dali logical agents (????).

[4] N. Ahmed, Y. Zhu, Early detection of atrial fibrillation based on ecg signals, Bioengineering 7 (2020) 16.

[5] M. Harris, J. Habetha, The myheart project: A framework for personal health care applications, in: 2007 Computers in Cardiology, 2007, pp. 137–140. doi:10.1109/CIC.2007.4745440.

[6] A. H. Jabber, A. Obied, A multi-agent system in e-health systemimplementing ebdi model, Turkish Journal of Computer and Mathematics Education 12 (2021) 2845–2859.

[7] L.-H. Juang, M.-N. Wu, Psoriasis image identification using k-means clustering with morphological processing, Measurement 44 (2011) 895–905.

[8] L. de Lauretis, T. Lombardi, S. Costantini, L. Clementini, An arduino-based device to detect dangerous audio noises (2021).

[9] L. De Lauretis, T. Lombardi, S. Costantini, Earsaver: A device to detect dangerous audio noises., in: AAI4H@ ECAI, 2020, pp. 4–7.

[10] S. Costantini, A. Tocchio, About declarative semantics of logic-based agent languages, in: M. Baldoni, U. Endriss, A. Omicini, P. Torroni (Eds.), Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Selected and Revised Papers, volume 3904 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 106–123.

[11] D. Piatov, S. Helmer, A. Dignös, F. Persia, Cache-efficient sweeping-based interval joins for extended allen relation predicates, The VLDB Journal (2021) 1–24.

[12] J. F. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (1983) 832–843. URL: https://doi.org/10.1145/182.358434. doi:10.1145/182.358434.

[13] S. Helmer, F. Persia, Iseql, an interval-based surveillance event query language, International Journal of Multimedia Data Engineering and Management (IJMDEM) 7 (2016) 1–21.

[14] J. Fayn, L. Conti, S. Fareh, P. Maison-Blanche, P. Nony, P. Rubel, Interactive and dynamic ECG analysis. Is it just an IDEA or a clinically relevant approach?, J Electrocardiol 29 Suppl (1996) 21–25.

[15] B. T. Fengler, W. J. Brady, C. U. Plautz, Atrial fibrillation in the wolff-parkinson-white syndrome: Ecg recognition and treatment in the ed, The American Journal of Emergency Medicine 25 (2007) 576–583. URL: https://www.sciencedirect.com/science/article/pii/S0735675706004505. doi:https://doi.org/10.1016/j.ajem.2006.10.017.

[16] F. Persia, S. Helmer, A framework for high-level event detection in a social network context via an extension of iseql, in: 2018 IEEE 12th International Conference on Semantic Computing (ICSC), 2018, pp. 140–147. doi:10.1109/ICSC.2018.00028.

Exploiting Probabilistic Trace Expressions for Decentralized Runtime Verification with Gaps

Davide Ancona[1,†], Angelo Ferrando[1,†] and Viviana Mascardi[1,*,†]

[1] *DIBRIS, University of Genova, Via Dodecaneso 35, Genova 16146, Italy*

Abstract

Multiagent Systems (MASs) are distributed systems composed by autonomous, reactive, proactive, heterogeneous communicating entities. In order to dynamically verify the behavior of such complex systems, a decentralized solution able to scale with the number of agents is necessary. When, for physical, infrastructural, or legal reasons, the monitor is not able to observe all the events emitted by the MAS, gaps are generated. In this paper we present a runtime verification decentralized approach to handle observation gaps in a MAS.

Keywords

Decentralized Runtime Verification, Probabilistic Trace Expressions, Observation Gaps, Multi-agent Systems, Agent Interaction Protocols

1. Introduction and Motivations

Distributed Runtime Verification (DRV) is a relatively new research sub-field aimed at designing fault-tolerant distributed algorithms that monitor other distributed algorithms, with the end goal of developing lightweight software systems that are more efficient that traditional verification techniques [1, 2]. The literature on DRV is almost limited [3, 4, 5, 6, 7, 8] and becomes even more limited when we consider DRV of a special kind of systems: multiagent systems (MASs [9]). In the MAS area, in fact, we are only aware of our own previous works [10, 11, 12].

Another sub-field which is raising more and more attention in the RV area concerns partial observability of the monitored events which can cause gaps in the event traces [13, 14, 15, 16]. Also in this case, when we consider MASs as the target system of the verification activity, we find very few works, all related with norm monitoring [17, 18].

This paper addresses the two issues above, decentralized runtime verification of partially observable systems, in a MAS context. The findings presented in this work can be

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

*Corresponding author.

†These authors contributed equally.

davide.ancona@unige.it (D. Ancona); angelo.ferrando@unige.it (A. Ferrando); viviana.mascardi@unige.it (V. Mascardi)

https://person.dibris.unige.it/ancona-davide/ (D. Ancona); https://angeloferrando.github.io/website/ (A. Ferrando); https://person.dibris.unige.it/mascardi-viviana/ (V. Mascardi)

0000-0002-6297-2011 (D. Ancona); 0000-0002-8711-4670 (A. Ferrando); 0000-0002-2261-9926 (V. Mascardi)

generalized and applied to other kinds of systems, but – for presentation purposes – we concentrate our investigation on MASs.

The main source of inspiration for our work is the paper by Stoller et al. [14], where the authors introduce *runtime verification with state estimation.* With respect to a more standard RV approach, they are interested in checking system executions (traces) containing *gaps.* A gap represents the absence of information in the trace of observed events and corresponds to an execution point where the monitor knows that the system emitted some event, but does not know which one. In offline RV gaps in the trace logs are due to the process of sampling observed events in order to reduce the monitoring overhead. Gaps can also be met in online RV, where the system behavior is analyzed while the system is running and problems with the infrastructure, privacy and legal issues that prevent the monitor to observe some kind of events, faults in the monitor observation capabilities, may generate gaps. Although the problems raised by online and offline RV with gaps share many similarities, the online setting is much more challenging. Each time a gap is perceived, the monitor must make guesses on the possible actual events that the gap represents and save all the states generated by these guesses. A possibly huge logical tree-like structure with states as nodes, and moves from states to states as edges, represents the open possibilities[1]. In offline RV, this logical tree-like structure can be explored following a depth-first search, requiring a limited amount of memory. If the RV takes place online, its exploration must follow a breadth-first strategy, with much more space needed to save the states, as the final trace of events is unknown and the levels of the structure are generated and explored at the same time. In order to cope with the state space explosion due to guesses in the online RV scenario, we propose to *decentralize* the monitoring activity.

RV decentralization is a very natural choice when the system under monitoring is a MAS, which is *distributed by definition*, and may improve *efficiency*, as the verification process can be spread on different machines improving performance; *scalability*, as under some conditions depending on the protocol [12, 10] it is possible to associate one monitor with each agent in the MAS, keeping under control the RV complexity even when the number of agents grows; *feasibility*, as for physical/logical/legal reasons one single monitor might not be able to observe all the events generated by the MAS.

The feature that is usually subject to verification (both static and dynamic) in a MAS is its *communicative behavior* [19, 20, 21, 22, 23, 24, 25, 26, 27]. With respect to [14], in this work we do not aim at verifying temporal properties. Rather, we want to check the conformance of the MAS actual communicative behavior to an Agent Interaction Protocol (AIP) that models the allowed interactions among agents, under the hypotesis that some interactions could not be observed. The research question we address is thus *how to evaluate the probability that a MAS satisfies an AIP, in the presence of gaps.*

In a recent paper [28] we introduced Probabilistic Trace Expressions (PTEs) and the theory behind them. In this work we take a more pragmatical perspective and we show

[1]In the remainder we will use the term “branch” to denote paths in this logical structure, and we will sometime use “states” meaning “the final states of all the possible branches”, when this does not generate confusion.

how to use PTEs for decentralized RV of AIPs within MASs with gaps.

2. Background

Probabilistic Trace Expressions. Trace expressions [29, 30, 31, 32, 33, 34, 35, 36, 37] are based on the notions of *event* and *event type*. We denote by $\mathcal{E}$ the fixed universe of events subject to monitoring. An event trace over $\mathcal{E}$ is a possibly infinite sequence of events in $\mathcal{E}$, and a trace expression over $\mathcal{E}$ denotes a set of event traces over $\mathcal{E}$. Trace expressions are built on top of event types (chosen from a set $\mathcal{ET}$), each specifying a subset of events in $\mathcal{E}$. A trace expression $\tau \in \mathcal{T}$ represents a set of possibly infinite event traces, and is defined on top of the following operators:
• ϵ (empty trace), denoting the singleton set $\{\epsilon\}$ containing the empty event trace ϵ.
• $\vartheta{:}\tau$ (*prefix*), denoting the set of all traces whose first event e matches the event type ϑ, and the remaining part is a trace of τ.
• $\tau_1{\cdot}\tau_2$ (*concatenation*), denoting the set of all traces obtained by concatenating the traces of τ_1 with those of τ_2.
• $\tau_1{\wedge}\tau_2$ (*intersection*), denoting the intersection of the traces of τ_1 and τ_2.
• $\tau_1{\vee}\tau_2$ (*union*), denoting the union of the traces of τ_1 and τ_2.
• $\tau_1|\tau_2$ (*shuffle*), denoting the set obtained by shuffling the traces of τ_1 with the traces of τ_2.

Trace expressions support recursion through cyclic terms expressed by finite sets of recursive syntactic equations, as supported by modern Prolog systems.

A probabilistic trace expression is a trace expression where event types have a probability associated with them [28], and its modeling and semantics are also implemented in Prolog (SWI-Prolog, see the code available here, https://vivianamascardi.github.io/Software/PTE.pl). PTEs are suitable to manage guesses in the presence of observation gaps; in order for this management to work, we assume that *each gap represents one single unobserved event.*

As an example, the probabilistic trace expression

$$\tau = e_1[0.2]{:}\tau_1{\vee}e_2[0.8]{:}(\tau_2|\tau_3)$$

represents the protocol where we can accept the event e_1 with probability 0.2, or, the event e_2 with probability 0.8. If we consume the event e_1, we go to the new state τ_1, while, if we consume e_2, we go to a state where we can have all possible interleaving of τ_2 and τ_3. If there is a gap in the monitoring activity and the monitor is not able to observe which event took place, it can nevertheless make its guesses which involve e_1 and e_2, associate a probability with each of them, and keep both possibilities.

Like a "normal" trace expression, a probabilistic trace expression τ can be seen as the current state of a protocol that started in some initial state τ_{init} and reached τ after n events $O_1 ... O_n$ took place. These events moved τ_{init} to τ through intermediate states τ_{q1}, τ_{q2}, ... , $\tau_{qn} = \tau$. If we denote with $\tau \overset{O}{\rightarrow} \tau'$ the transition from state τ to state τ' due to the event O taking place and being observed, we may write

$\tau_{init} \overset{O_1}{\rightarrow} \tau_{q1} \overset{O_2}{\rightarrow} \tau_{q2} \overset{O_3}{\rightarrow} \tau_{q3} ... \overset{O_n}{\rightarrow} \tau_{qn}$, where $\tau_{qn} = \tau$.

In order to properly manage probabilities, it is convenient to associate with τ – in an explicit and easily computable way – the probability of the protocol to have reached τ starting from τ_{init} and having observed $O_1...O_n$.

We define a "probabilistic trace expression state" the triple consisting of a trace expression τ, a sequence of events $O_1...O_n$ observed before reaching τ, and the probability π_τ that the protocol reached τ. We represent the state with the notation $\langle \tau, \pi_{tr}, O_1...O_n \rangle$.

Decentralized MAS Monitoring with DecAMon. In [10] we presented the DecAMon algorithm to decentralize agent interaction protocols modeled using trace expressions. There, we defined the notion of "monitoring safe" partition. A partition can be used to drive the distribution process. To decentralize the monitoring activity, we project the global AIP onto each subset of agents belonging to the partition, where by "projection" we mean that we maintain only the interactions involving agents in the chosen subset. In general, not all the partitions can be used for the RV decentralization. A partition that can be used to decentralize the RV of a protocol is called "monitoring safe" and the algorithm presented in [10] generates all the monitoring safe partitions for a given AIP.

Since under the conditions considered in this paper we may observe gaps, we could not have only one single state representing the current situation of the protocol, like it happens in our previous works; instead, we have to maintain all the states that may be possibly reached "via the gaps". As already anticipated, each state can be represented as a tuple $\langle \tau, \pi, evs \rangle$, where τ is the PTE representing the current state of the protocol and π is the joint probability that the sequence of events evs is compliant with τ [28].

Let us name M_0 the set of possible initial states of the monitor (as there may be more than one). The number 0 stands for the $0th$ iteration, since at the beginning we have not consumed any event yet. We can first run DecAMon on the global AIP to find a good set of monitoring safe partitions and, after that, we can use one of them to project the τs in M_0 onto the subsets of the agents. Once we have obtained the distributed versions of the initial τs via projection, we can generate one monitor for each partition, and decentralize the RV.

The combination of decentralization and lack of information calls for a synchronized management of gaps. Since each monitor has a different state representing its current protocol evolution, when there is an observation gap, each monitor can have different opinions about which are the correct events that might suitably "fill the gap". The local perspectives can be compared and used by the monitors to cut wrong guesses, and hence wrong states, on the basis of distributed knowledge. Despite the overhead due to synchronization, this approach may dramatically improve performance, as discussed in the next sections.

3. Handling Gaps in Decentralized RV

Gaps represent lack of information, thus a point (or points) in the event trace where the monitor does not know what event had been actually generated by the system under monitoring. In the remainder we will write that "gaps can be observed", in the sense that a monitor can realize that something went wrong and that an event was generated by the system, and not correctly observed. We also assume that, in a decentralized setting, when one monitor "observes a gap", all the monitors "observe a gap" as well. If this gap does not involve the sub-system monitored by a monitor M, the trace observed by M will contain $gap(none)$: this notation means – from M's point of view – "I am aware that some event was generated by some component of the system that I am not in charge of, and that the event was not correctly observed". From a technical viewpoint, this could be obtained by forcing one monitor to inform the others when it observes a gap. This would require some shared clock among the monitors as, in order for our algorithm to work, the gap must take place at the same time for all the monitors hence raising clock synchronization issues. Given that these issues are well known and well studied in distributed systems [38], we leave them out of our investigation. Being well studied does not mean to be easy to face. Indeed, we are aware that the need of observing all the gaps at the same time in a decentralized setting, represents a serious limitation of our framework and we are working towards alternative, and more feasible, solutions.

When a centralized monitor observes a gap, since it is the only monitor checking the event trace w.r.t. the AIP specification, it can make guesses on what the gap is and reason on its own guesses, eventually tagging some of them as wrong due to successive observations. When there are many monitors, each one monitoring a subset of the agents, and hence a sub-protocol of the global AIP, each monitor can still suppose what the observed gap is, but the reasoning on its suppositions must be shared with the others. This sharing phase among the monitors is crucial, because it allows them to cut wrong branches on the basis of what other monitors suppose, or what they are fully sure of.

Let us consider two monitors m_1 and m_2 that observe a gap. Given that the protocols driving the two monitors are different, although being derived via projection from the same global protocol, m_1 might suppose that the events admissible for filling the observed gap are e_1 and e_2, while m_2 could instead suppose that admissible events are e_2 and e_3. Both m_1 and m_2 must keep track of these possibilities in their local knowledge bases, and – so far – they do not need to share they guesses.

Let us now suppose that in the current state of m_1, in the branch where e_1 was supposed to have taken place, the only successive possible event is e_4, while in the branch for e_2 the only possible event is e_5. If, after the gap, m_1 observes e_5, it can cut the branch where the gap was associated with e_1, because e_5 would not be allowed after e_1. The gap before e_5, that could be filled in principle by e_1 and e_2, becomes bound - "without any doubt"[2] - to e_2. After having found the right value for the gap and cut one branch, m_1 informs m_2 allowing it to cut the branch where the value for that gap was guessed to be

[2]Modulo the assumption that observed events are compliant with the foreseen protocol. Gaps may inevitably generate false negatives. In this case, m_1 assumes that the gap was e_2 because this would be consistent with the successive observation of e_5 and with the protocol to be respected. If the gap were any other event, a protocol violation would have taken place and m_1 should have raised a protocol monitoring exception. Depending on the protocol, the violation could be recognized later on, or never. Suppose for example an infinite protocol where only as are allowed. A gap will be necessarily filled with a even if the actual event was b, and if the successive observed events are all as, the violation will never be discovered.

e_3. In this way, both m_1 and m_2 can continue the verification process supposing that the unobserved event represented by the gap was e_2, with some given probability due to the probability associated with e_2 in the PTE modelling the protocol.

Before presenting the decentralized monitoring algorithm, we make some considerations on the kind of gaps a monitor can observe. So far, we considered generic events. This is correct and consistent with the general approach presented in [28], but in a MAS scenario where PTEs model agent interaction protocols we can be more specific. In this scenario, in fact, the universe of events is $Msgs$, namely the universe of the possible messages among agents. Such special events can be represented as $a_1 \overset{c}{\Longrightarrow} a_2$, meaning that agent a_1 sends a message to a_2 with content c. Since messages are composed by (at least) three mandatory components, sender, receiver and content, there can be many partially instantiated gaps such as:

- $gap(a_1 \Longrightarrow a_2)$, where the content of the message is unknown;
- $gap(\_ \overset{m}{\Longrightarrow} a_2)$, where the sender is unknown;
- $gap(a_1 \overset{m}{\Longrightarrow} \_)$, where the receiver is unknown.

Although, for sake of clarity, in the sequel we consider gaps where neither the sender, nor the receiver, nor the content are known (total absence of information), all the combinations of "information holes" are possible, and partially instantiated gaps may be exploited to reduce branches due to guesses. The algorithm presented in the next section can be easily adjusted to take partially instantiated gaps into account.

Synchronizing Decentralized Gaps Management. We present the algorithm used by the decentralized monitors to synchronize the gaps management, in order to cut useless branches and check the compliance of interactions with the protocol. When an event is generated by the system, two different situations can take place.

Case 1: The event is not a gap

If the event is not a gap, each monitor that observed it can use the event for updating its local state(s). If some branches have been removed as in the previous example involving m_1 and m_2, the monitor has to inform the other monitors of the associations between gaps and events that are not admissible any longer. This phase can be reiterated until all the monitors have cut all the possible wrong brnches, and have nothing more to say. After this synchronizations stage, the monitoring process continues in the normal way.

Case 2: The event is a gap

To keep the presentation simple, we assume that gaps are observed by all the monitors at the same time. Each monitor guesses the events admissible to fill the gap, according to its local states. If the gap is partially instantiated (some of its components were correctly observed, like the sender, or the content, or both), the monitor can use this information to reduce the set of possible candidate events.

The two cases can be seen as a *reduce* and *extend* stages, respectively. When the monitor observes a fully instantiated event it can invalidate zero, one or more branches. If the invalid branches contain gaps, the monitor can also invalidate the associations between these gaps and the guessed events, and can allow the other monitors to invalidate these associations as well via communication. On the other hand, observation of gaps generates as many branches as the events that, according to the AIP, could fill the gap. We can formalize this intuition in the following way.

Given M_0 as the set of global states $\{\langle \tau_1, \pi_1, []\rangle, ..., \langle \tau_n, \pi_n, []\rangle\}$.

1. Distribute M_0 with respect to a given partition $P = \{\{ags_1\}, ..., \{ags_{np}\}\}$, projecting the states onto subsets of the agents involved (the function Π projects an AIP τ onto a set of agents ags removing all the events whose sender and receiver do not belong to ags), obtaining

$$M_{0,\{ags_1\}} = \{\langle \Pi(\tau_1, \{ags_1\}), \pi_1, []\rangle, ..., \langle \Pi(\tau_n, \{ags_1\}), \pi_n, []\rangle\}$$
$$...$$
$$M_{0,\{ags_{np}\}} = \{\langle \Pi(\tau_1, \{ags_{np}\}), \pi_1, []\rangle, ..., \langle \Pi(\tau_n, \{ags_{np}\}), \pi_n, []\rangle\}$$

2. Each monitor observes only the event messages involving the agents belonging to its set ags_i:
   a) if the event message is a gap, the monitor guesses what it could be and generates as many states as the possible events (*extend*);
   b) if the event message is ground, the monitor can cut branches, and in this case it communicates with other monitors the gap values that are no longer admissible (*reduce*).
3. If, after observation of an event or because of information received from other monitors, the set of possible current states for a monitor m becomes empty, m stops the monitoring process, informs all the other monitors, and they also stop monitoring. The absence of possible current states for a monitor is due to a protocol violation that took place, preventing at least one monitor to move a further step. So, the system checked does not satisfy the agent interaction protocol and the associated probability is 0.
4. Else,
   a) if there are no events left to analyze, the monitoring process ends and the resulting probability is evaluated (see after how);
   b) else, repeat from step 2.

To be clearer, in step 2, given the current event message, each monitor queries its current state following the PTE operational semantics presented in [28] in order to check if the event message is admissible or not. In the updating phase, the monitors inform the others trying to cut not admissible branches.

If the monitoring process ends without violations detected and there are no more events left to analyze, each monitor stops with at least one admissible branch. Each monitor states its own evaluation of the probability that the system's behavior satisfies the agent interaction protocol. This probability can be computed summing up all the joint probabilities contained in all the final states, corresponding to the last nodes of the admissible branches. This leads to having one estimated value for each monitor: we can adopt different strategies to summarize the final, and global, one, such as taking the smallest (largest) value among all those estimated by all the monitors, or a weighted means where weights model each monitor's trustability, or other domain-dependent strategies.

4. Example

We present a simple example helping us to show how the *extend* and *reduce* steps work. We consider a MAS involving four agents: $\{alice, bob, charlie, dave\}$. The set of events of our interest is the set of messages that these agents can use to communicate with each other.

Given the PTE

$$\tau = \tau_1 \vee \tau_2$$
$$\tau_1 = alice \overset{msg_1}{\Longrightarrow} bob[0.7]{:}(bob \overset{msg_2}{\Longrightarrow} charlie[0.6]{:}\tau_1 | bob \overset{msg_3}{\Longrightarrow} dave[0.4]{:}\epsilon)$$
$$\tau_2 = alice \overset{msg_4}{\Longrightarrow} dave[0.3]{:}(charlie \overset{msg_5}{\Longrightarrow} dave[0.3]{:}\epsilon | bob \overset{msg_3}{\Longrightarrow} dave[0.7]{:}\tau_2)$$

We decentralize τ on each single agent, obtaining[3]:

$$M_{0,\{alice\}} = \{\langle \Pi(\tau, \{alice\}), 1, [] \rangle\} = \{\langle \tau_{alice}, 1, [] \rangle\}$$
$$M_{0,\{bob\}} = \{\langle \Pi(\tau, \{bob\}), 1, [] \rangle\} = \{\langle \tau_{bob}, 1, [] \rangle\}$$
$$M_{0,\{charlie\}} = \{\langle \Pi(\tau, \{charlie\}), 1, [] \rangle\} = \{\langle \tau_{charlie}, 1, [] \rangle\}$$
$$M_{0,\{dave\}} = \{\langle \Pi(\tau, \{dave\}), 1, [] \rangle\} = \{\langle \tau_{dave}, 1, [] \rangle\}$$

where

$$\tau_{alice} = \tau_{1_{alice}} \vee \tau_{2_{alice}}$$
$$\tau_{1_{alice}} = alice \overset{msg_1}{\Longrightarrow} bob[0.7]{:}\tau_{1_{alice}}$$
$$\tau_{2_{alice}} = alice \overset{msg_4}{\Longrightarrow} dave[0.3]{:}\tau_{2_{alice}}$$
$$\tau_{bob} = \tau_{1_{bob}} \vee \tau_{2_{bob}}$$
$$\tau_{1_{bob}} = alice \overset{msg_1}{\Longrightarrow} bob[0.7]{:}(bob \overset{msg_2}{\Longrightarrow} charlie[0.6]{:}\tau_1 | bob \overset{msg_3}{\Longrightarrow} dave[0.4]{:}\epsilon)$$
$$\tau_{2_{bob}} = bob \overset{msg_3}{\Longrightarrow} dave[0.7]{:}\tau_{2_{bob}}$$
$$\tau_{charlie} = \tau_{1_{charlie}} \vee \tau_{2_{charlie}}$$
$$\tau_{1_{charlie}} = bob \overset{msg_2}{\Longrightarrow} charlie[0.6]{:}\tau_{1_{charlie}}$$
$$\tau_{2_{charlie}} = charlie \overset{msg_5}{\Longrightarrow} dave[0.3]{:}\tau_{2_{charlie}}$$
$$\tau_{dave} = \tau_{1_{dave}} \vee \tau_{2_{dave}}$$

[3]The initial probability of each state is 1, since we do not want to influence the probability evaluation process (multiplication of probabilities).

$$\tau_{1_{dave}} = bob \overset{msg_3}{\Longrightarrow} dave[0.4]{:}\epsilon$$

$$\tau_{2_{dave}} = alice \overset{msg_4}{\Longrightarrow} dave[0.3]{:}(charlie \overset{msg_5}{\Longrightarrow} dave[0.3]{:}\epsilon | bob \overset{msg_3}{\Longrightarrow} dave[0.7]{:}\tau_{2_{dave}})$$

Let us suppose that the monitors observe a *gap* now. Each monitor moves to a new set of states corresponding to the possible values for the *gap*.

$$M_{0,\{alice\}} \overset{gap}{\rightarrow} \{\langle \tau_{1_{alice}}, 0.7, [gap(alice \overset{msg_1}{\Longrightarrow} bob)]\rangle,$$
$$\langle \tau_{2_{alice}}, 0.3, [gap(alice \overset{msg_4}{\Longrightarrow} dave)]\rangle,$$
$$\langle \tau_{alice}, 1, [gap(none)]\rangle\} = M_{1,\{alice\}}$$
$$M_{0,\{bob\}} \overset{gap}{\rightarrow} \{$$
$$\langle (bob \overset{msg_2}{\Longrightarrow} charlie[0.6]{:}\tau_1 | bob \overset{msg_3}{\Longrightarrow} dave[0.4]{:}\epsilon), 0.7, [gap(alice \overset{msg_1}{\Longrightarrow} bob)]\rangle,$$
$$\langle \tau_{2_{bob}}, 0.7, [gap(bob \overset{msg_3}{\Longrightarrow} dave)]\rangle,$$
$$\langle \tau_{bob}, 1, [gap(none)]\rangle\} = M_{1,\{bob\}}$$
$$M_{0,\{charlie\}} \overset{gap}{\rightarrow} \{$$
$$\langle \tau_{1_{charlie}}, 0.6, [gap(bob \overset{msg_2}{\Longrightarrow} charlie)]\rangle,$$
$$\langle \tau_{2_{charlie}}, 0.3, gap(charlie \overset{msg_5}{\Longrightarrow} dave)\rangle,$$
$$\langle \tau_{charlie}, 1, [gap(none)]\rangle\} = M_{1,\{charlie\}},$$
$$M_{0,\{dave\}} \overset{gap}{\rightarrow} \{$$
$$\langle \epsilon, 0.4, gap(bob \overset{msg_3}{\Longrightarrow} dave)\rangle,$$
$$\langle (charlie \overset{msg_5}{\Longrightarrow} dave[0.3]{:}\epsilon | bob \overset{msg_3}{\Longrightarrow} dave[0.7]{:}\tau_{2_{dave}}), 0.3, gap(alice \overset{msg_4}{\Longrightarrow} dave)\rangle,$$
$$\langle \tau_{dave}, 1, [gap(none)]\rangle\} = M_{1,\{dave\}}$$

Since they observed a *gap*, the monitors do not know what the actual event was. Because of this, they have to generate more branches, where each branch represents a possible value for the gap. This is the *extend* step.

Let us now suppose that the monitors observe event msg_2. Since msg_2 is a ground event, everything is known about it, in particular the monitors know that its sender is *bob* and its receiver is *charlie*. Since the monitors observe only the gaps and the events that involve the agents in the partition they are in charge for, the only monitors that observe msg_2 are $M_{1,\{bob\}}$ and $M_{1,\{charlie\}}$.

By consuming msg_2, the first iteration of the algorithm leads to:

$$M_{1,\{bob\}} \overset{bob \overset{msg_2}{\Longrightarrow} charlie}{\rightarrow} \{$$
$$\langle \tau_1 | bob \overset{msg_3}{\Longrightarrow} dave[0.4]{:}\epsilon, 0.42, [gap(alice \overset{msg_1}{\Longrightarrow} bob), bob \overset{msg_2}{\Longrightarrow} charlie]\rangle$$
$$\} = M_{2,\{bob\}}$$

$$M_{1,\{charlie\}} \overset{bob \overset{msg_2}{\Longrightarrow} charlie}{\rightarrow} \{$$
$$\langle \tau 1_{charlie}, 0.36, [gap(bob \overset{msg_2}{\Longrightarrow} charlie), bob \overset{msg_2}{\Longrightarrow} charlie] \rangle,$$
$$\langle \tau 1_{charlie}, 0.6, [gap(none), bob \overset{msg_2}{\Longrightarrow} charlie] \rangle$$
$$\} = M_{2,\{charlie\}}$$

It is interesting to analyze what happened in $M_{2,\{bob\}}$, where the *reduce* step took place. In fact, the ground event msg_2 makes the other two branches not valid anymore. More in detail, the second branch was $\langle \tau 2_{bob}, 0.7, [gap(msg_3)] \rangle$, and $\tau 2_{bob}$ does not accept the event msg_2 and cannot move to a new state. In the same way, the PTE in the third branch $\langle \tau_{bob}, 1, gap(none) \rangle$ is τ_{bob}, and τ_{bob} cannot accept the event msg_2 either. Even though this information seems important for monitor $M_{2,\{bob\}}$ only, it is actually of interest also for the other monitors. In fact, it allows all of them to know "without any doubt" that the only event that can be associated with the first gap is msg_1, since it is the gap value associated with the only possible branch of $M_{2,\{bob\}}$. The monitor $M_{2,\{bob\}}$ can inform the other monitors that the only admissible value for the gap is msg_1. The monitors' new states become:

$$M_{2,\{charlie\}} = \{ \langle \tau 1_{charlie}, 0.6, [gap(none), bob \overset{msg_2}{\Longrightarrow} charlie] \rangle \}$$
$$M_{1,\{alice\}} = \{ \langle \tau 1_{alice}, 0.7, [gap(alice \overset{msg_1}{\Longrightarrow} bob)] \rangle \}$$
$$M_{1,\{dave\}} = \{ \langle \tau_{dave}, 1, [gap(none)] \rangle \}$$

This example shows how the knowledge of a monitor can have a positive impact on the knowledge of the other monitors. In general, this positive impact can be obtained any time one monitor discovers that one branch is no longer valid and can hence invalidate the associations of events with gaps therein. This information may trigger many communication iterations among the monitors, because, when one monitor is updated it can also "invalidate one branch" and the related gap-events associations, and may need to inform the others of some association which is no longer possible. In the previous example, one single iteration was enough.

As we already anticipated, the proposed approach may lead to false negatives, due to an optimistic approach of the monitors that stubbornly assume that observed events are compliant with the protocol, if there is just one possibility left to make such an assumption. Also in this example, the monitors gave the correctness of the ground event msg_2 (the second event observed) for granted. But let us suppose that the actual event masked by the *gap* was not msg_1, but msg_4, and that the successive message msg_2 was sent from *bob* to *charlie* by mistake and did not comply with the protocol. In this scenario, since the monitors do not know for sure what the first *gap* was, it is reasonable to consider msg_2 a valid message and hence cut the branch where the gap has been supposed to be msg_4. This is a problem intrinsically related to the state estimation approach, since until it is acceptable to observe an event in a state, the monitors keep track of the related branch. Only when a monitor, observing an event, loses all its branches it can conclude

Table 1
Average time of the centralized and decentralized algorithms; “sh. PTE” stands for “shuffled sub-PTE”.

# sh. PTEs	# agents for sh. PTE	# operations for sh. PTE	Centralized [sec]	Decentralized [sec]
10	10	20	6.64	1.26
10	10	15	8.26	1.04
10	5	20	9.85	1.49
10	5	15	9.92	1.28
10	15	15	14.86	1.23
10	5	10	18.35	1.08
10	15	10	20.25	1.61
10	10	10	29.59	1.98
15	5	15	93.34	2.73
15	15	10	116.61	3.56
10	15	20	126.31	25.32
15	10	10	283.70	4.14
15	5	10	349.30	2.23
20	10	10	355.90	3.99
15	5	20	363.67	5.83
20	5	15	558.59	9.28
20	5	20	801.37	7.82
15	20	10	952.43	12.36
20	5	10	1223.85	10.64
20	15	10	1340.29	9.57
20	20	10	1727.26	2.89

that a protocol violation took place because some wrong assumption on gaps – confirmed by successive observations – had been made in the past. This delay in the error detection, which could also be infinite, can be reduced introducing a threshold on the probability that a branch must have to be considered valid. In this way, if after observing an event the probability associated with a branch becomes lower than a chosen threshold, the monitor can cut that branch and make error detection possibly quicker.

5. Experimental results

In our experiments we have considered the four following features:

1. the number of agents involved in the MAS we want to verify at runtime;
2. the number of *shuffled sub-PTEs* due to shuffle operators $|$ in the AIP: we name shuffled sub-PTE each portion of the PTE composed via a $|$, so for example $\tau_3 = alice \overset{msg_1}{\Longrightarrow} bob[0.7]{:}\epsilon \mid bob \overset{msg_3}{\Longrightarrow} dave[0.4]{:}\epsilon$ consists of 2 *shuffled sub-PTEs*; we point out that when decentralizing the monitoring, we can associate one different monitor with each shuffled sub-PTE, as shuffled sub-PTE are independent one from the other and can be monitored in a fully decentralized way;
3. the number of operators for each shuffled sub-PTE in the AIP;
4. the number of gaps contained in the analyzed traces.

In Table 1, we report the results of our experiments. For each row, we keep the number of shuffled sub-PTE, agents and operators fixed, while we change the length of the traces and the percentage of gaps inside each trace. For each row we executed many different runs and we have measured the total time required for recognizing the set of 300 randomly generated traces. We changed the number of gaps contained inside the traces and we tested both the centralized [28] and the decentralized algorithms. In the following, we reported the graphics obtained from such executions.

Concerning the figures, *the traces used in our experiments contain only gaps* (namely, we run experiments in the worst possible scenario), so the algorithm makes only expansions and never reductions. We chose traces with only gaps to stress the algorithms as much as possible. In real scenarios gaps should be the exceptions, and perfectly observable events the norm.

In Figures 1 and 2, both the centralized and the decentralized algorithms seem to show linear complexity with respect the number of the agents involved, even if the decentralized algorithm has better performances.

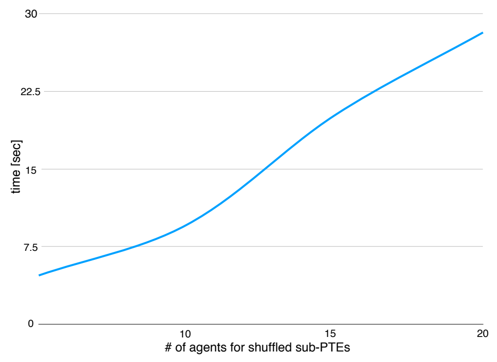


Figure 1: Centralized algorithm: changing number of agents.

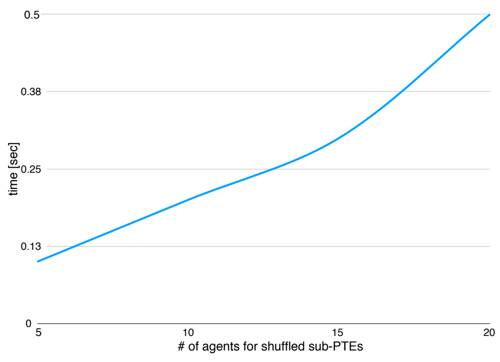


Figure 2: Decentralized algorithm: changing number of agents.

In Figures 3 and 4, we can observe that the complexity of the centralized algorithm seems to grow in a quadratic way, while the decentralized one seems to grows linearly. This can be explained by the decentralization of the monitoring of shuffled sub-PTEs, as if we add one operator to each shuffled sub-PTE, the monitor in charge for that shuffled sub-PTE will need to manage one more operator only, whereas the centralized monitor will cope with as many new operators as the shuffled sub-PTEs in the trace expression. We point out that we use "seems to" to reflect that the complexities emerging from the figures have not been computed on the basis of the algorithm, but have been estimated on the basis of the experiments, and the behaviour in situations involving a limited number of agents, operators, shuffled sub-PTEs, might not be the actual asymptotic behaviour of the algorithm.

In Figures 5 and 6, we can appreciate the real advantages of decentralization, as – from the figures – it seems that we have an exponential complexity for the centralized algorithm and a pseudo-quadratic complexity for the decentralized one. We emphasise that in the decentralized case (Figure 6) we were able to run experiments with 40 shuffled sub-PTEs, while in the centralized case we had to stop with half shuffled sub-PTEs, and with an execution time hundred times higher. The number of shuffled sub-PTEs is indeed the feature which most impacts the algorithms performance, and this in not a surprise; intuitively, when we add a new shuffled sub-PTE we have to interleave it with all the already existent shuffled sub-PTEs. In the centralized case, this brings to a

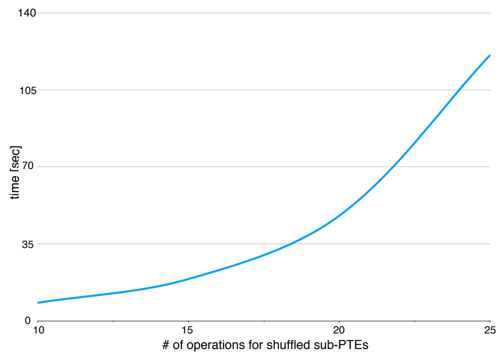


Figure 3: Centralized algorithm: changing number of operators.

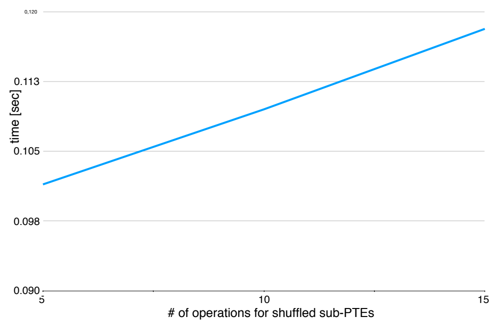


Figure 4: Decentralized algorithm: changing number of operators.

state explosion, while in the decentralized one, since we can decentralize the monitoring of each shuffled sub-PTEs, we simply have to add a new monitor. In this way, we can avoid the state explosion, even if the presence of a new monitor increases the exchange of messages among the monitors needed to synchronize information about gaps.

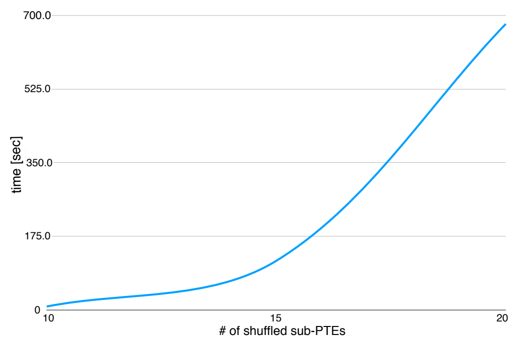


Figure 5: Centralized algorithm: changing number of shuffled sub-PTEs.

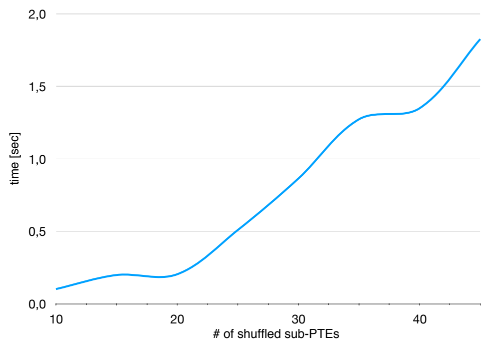


Figure 6: Decentralized algorithm: changing number of shuffled sub-PTEs.

So far, our experiments are simulated: although trace expressions and the RML language[4] that spun from them [37] have been adopted to model and dynamically verify behavioural patterns involving interaction among Jason and JADE agents [39], robotic systems [40], IoT systems [41] and programs developed in Node.js and Node-RED [37], their probabilistic extension, PTEs, was not tested on real setting yet.

The code for running our experiments can be found in the PTE repository on GitHub, https://vivianamascardi.github.io/Software/PTE.pl.

[4]https://rmlatdibris.github.io/, accessed on June 2022.

By calling the `generate_mas` goal, we can generate one simulated PTE with some features, for example `generate_mas(5, 3, 4, Partition, T)` unifies `T` with a PTE with 5 branches, 3 randomly generated agents involved in each branch, and 4 randomly selected operators (such as shuffle, union, etc) for each branch, and `Partition` with a partition of `T` into sub-PTEs to be monitored in a decentralized way. Then, the `T` and `Partition` variables unified with ground temrs can be used as arguments of the `create_output_file` goal that generates a csv file containing all the data needed for making the experiments presented in this section.

In `create_output_file(ID, T, MaxLength, NTests, MinProbNoise, MaxProbNoise, MinProbMsgNoise, MaxProbMsgNoise, Partition)`, `ID` is the csv file name to be generated, `T` and `Partition` must be unified with a ground PTE and with a partition into sub-PTEs, respectively, by calling `generate_mas`, `MaxLength` is the maximum length of the trace to analyze, `NTests` is the number of tests to be repeated, to have more robust and reliable results, `MinProbNoise` and `MaxProbNoise` define the probability range to have gaps in the generated and analyzed trace, `MinProbMsgNoise` and `MaxProbMsgNoise` define the probability range to have gaps in the observed message.

6. Conclusions and Future Work

In this paper we presented a distributed approach to runtime verification where we may lack some pieces of information about observed events. With respect to standard runtime verification, the state estimation approach allows us to be more reliable, especially in scenarios where partial or total absence of information is frequent.

For the sake of clarity, we considered only totally uninstantiated gaps. This choice has been made to make the development of monitors easier. Naturally, the presence of part of information about the event could be used by the monitors in order to cut useless branches. We will extend our implementation to cope with partially instantiated gaps.

Another future work will be to consider a threshold in order to cut branches that are unreasonable to maintain, as the probability to be correct is too low. Fixed a threshold, a monitor will be able to remove all the branches with a joint probability associated with them lower than the chosen threshold. This will bring the advantage of anticipating the error detection and to prune useless branches related to unreasonable possibilities.

References

[1] B. Bonakdarpour, P. Fraigniaud, S. Rajsbaum, C. Travers (Eds.), Bertinoro Seminar on Distributed Runtime Verification, May 2016, Available from `http://www.labri.fr/perso/travers/DRV2016/`, 2016.

[2] B. Bonakdarpour, P. Fraigniaud, S. Rajsbaum, C. Travers, Challenges in fault-tolerant distributed runtime verification, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications: 7th International Symposium, ISoLA 2016. Proceedings, Part II, Springer, 2016, pp. 363–370.

[3] P. Fraigniaud, S. Rajsbaum, M. Roy, C. Travers, The opinion number of set-agreement, in: M. K. Aguilera, L. Querzoni, M. Shapiro (Eds.), Principles of Distributed Systems: 18th International Conference, OPODIS 2014. Proceedings, Springer, 2014, pp. 155–170.

[4] P. Fraigniaud, S. Rajsbaum, C. Travers, On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems, in: B. Bonakdarpour, S. A. Smolka (Eds.), Runtime Verification: 5th International Conference, RV 2014. Proceedings, Springer, 2014, pp. 92–107.

[5] M. Herlihy, Wait-free synchronization, ACM Trans. Program. Lang. Syst. 13 (1991) 124–149.

[6] M. Mostafa, B. Bonakdarpour, Decentralized runtime verification of LTL specifications in distributed systems, in: Parallel and Distributed Processing Symposium, IEEE International Conference, IPDPS 2015. Proceedings, 2015, pp. 494–503.

[7] Y. Falcone, T. Cornebize, J.-C. Fernandez, Efficient and generalized decentralized monitoring of regular languages, in: E. Ábrahám, C. Palamidessi (Eds.), Formal Techniques for Distributed Objects, Components, and Systems: 34th IFIP WG 6.1 International Conference, FORTE 2014. Proceedings, Springer, 2014, pp. 66–83.

[8] E. Bartocci, Sampling-based decentralized monitoring for networked embedded systems, in: HAS, volume 124 of *EPTCS*, 2013, pp. 85–99.

[9] M. Wooldridge, N. R. Jennings, Intelligent agents: theory and practice, Knowledge Eng. Review 10 (1995) 115–152.

[10] A. Ferrando, D. Ancona, V. Mascardi, Decentralizing MAS monitoring with decamon, in: K. Larson, M. Winikoff, S. Das, E. H. Durfee (Eds.), Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017, ACM, 2017, pp. 239–248. URL: http://dl.acm.org/citation.cfm?id=3091164.

[11] D. Ancona, D. Briola, A. Ferrando, V. Mascardi, MAS-DRiVe: a practical approach to decentralized runtime verification of agent interaction protocols, in: C. Santoro, F. Messina, M. D. Benedetti (Eds.), From Objects to Agents, 17th Workshop, WOA 2016. Proceedings, volume 1664 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 35–43. URL: http://ceur-ws.org/Vol-1664.

[12] D. Ancona, A. Ferrando, L. Franceschini, V. Mascardi, Coping with bad agent interaction protocols when monitoring partially observable multiagent systems, in: PAAMS, volume 10978 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 59–71.

[13] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, S. A. Smolka, Model repair for probabilistic systems, in: TACAS, volume 6605 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 326–340.

[14] S. D. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. A. Smolka, E. Zadok, Runtime verification with state estimation, in: RV, volume 7186 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 193–207.

[15] Y. Joshi, G. M. Tchamgoue, S. Fischmeister, Runtime verification of LTL on lossy traces, in: SAC, ACM, 2017, pp. 1379–1386.

[16] R. Babaee, A. Gurfinkel, S. Fischmeister, $\mathcal{P}revent:$ A predictive run-time verification

framework using statistical learning, in: SEFM, volume 10886 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 205–220.

[17] N. Criado, J. M. Such, Norm monitoring under partial action observability, IEEE Trans. Cybernetics 47 (2017) 270–282.

[18] N. Criado Pacheco, Resource-bounded norm monitoring in multi-agent systems, Journal Artificial Intelligence Research (2018) 1.

[19] M. Baldoni, C. Baroglio, F. Capuzzimati, A commitment-based infrastructure for programming socio-technical systems, ACM Trans. Internet Techn. 14 (2014) 23:1–23:23. doi:10.1145/2677206.

[20] M. Baldoni, C. Baroglio, F. Capuzzimati, R. Micalizio, Exploiting social commitments in programming agent interaction, in: Proc. of PRIMA 2015, volume 9387 of *LNCS*, Springer, 2015, pp. 566–574.

[21] A. K. Chopra, M. P. Singh, Cupid: Commitments in relational algebra, in: Proc. of AAAI 2015, AAAI Press, 2015, pp. 2052–2059.

[22] M. Winikoff, W. Liu, J. Harland, Enhancing commitment machines, in: Proc. of DALT 2004, Revised Selected Papers, volume 3476 of *LNCS*, Springer, 2004, pp. 198–220.

[23] P. Yolum, M. P. Singh, Commitment machines, in: Proc. of ATAL 2001, Revised Papers, volume 2333, Springer, 2002, pp. 235–247.

[24] A. K. Chopra, S. Christie, M. P. Singh, Splee: A declarative information-based language for multiagent interaction protocols, in: Proc. of AAMAS 2017, ACM, 2017, pp. 1054–1063.

[25] M. P. Singh, Information-driven interaction-oriented programming: BSPL, the blindingly simple protocol language, in: Proc. of AAMAS 2011, IFAAMAS, 2011, pp. 491–498.

[26] M. Winikoff, N. Yadav, L. Padgham, A new hierarchical agent protocol notation, Autonomous Agents and Multi-Agent Systems 32 (2018) 59–133.

[27] D. Ancona, A. Ferrando, V. Mascardi, Improving flexibility and dependability of remote patient monitoring with agent-oriented approaches, Int. J. Agent Oriented Softw. Eng. 6 (2018) 402–442. URL: https://doi.org/10.1504/IJAOSE.2018.096422. doi:10.1504/IJAOSE.2018.096422.

[28] D. Ancona, A. Ferrando, V. Mascardi, Mind the gap! Runtime verification of partially observable MASs with probabilistic trace expressions, in: D. Baumeister, J. Rothe (Eds.), The 19th European Conference on Multi-Agent Systems (EUMAS 2022), Springer, 2022.

[29] D. Ancona, S. Drossopoulou, V. Mascardi, Automatic generation of self-monitoring MASs from multiparty global session types in Jason, in: Declarative Agent Languages and Technologies X - 10th International Workshop, DALT 2012, Valencia, Spain, June 4, 2012, Revised Selected Papers, 2012, pp. 76–95.

[30] D. Ancona, M. Barbieri, V. Mascardi, Constrained global types for dynamic checking of protocol conformance in multi-agent systems, in: SAC, ACM, 2013, pp. 1377–1379.

[31] D. Briola, V. Mascardi, D. Ancona, Distributed runtime verification of JADE multiagent systems, in: IDC, volume 570 of *Studies in Computational Intelligence*, Springer, 2014, pp. 81–91.

[32] D. Ancona, D. Briola, A. Ferrando, V. Mascardi, Global protocols as first class entities for self-adaptive agents, in: AAMAS, ACM, 2015, pp. 1019–1029.

[33] D. Ancona, V. Bono, M. Bravetti, J. Campos, G. Castagna, P. Deniélou, S. J. Gay, N. Gesbert, E. Giachino, R. Hu, E. B. Johnsen, F. M. et al., Behavioral types in programming languages, Foundations and Trends in Programming Languages 3 (2016) 95–230.

[34] D. Ancona, A. Ferrando, V. Mascardi, Comparing trace expressions and linear temporal logic for runtime verification, in: Theory and Practice of Formal Methods, volume 9660 of *LNCS*, 2016, pp. 47–64.

[35] D. Ancona, A. Ferrando, V. Mascardi, Parametric runtime verification of multiagent systems, in: AAMAS, ACM, 2017, pp. 1457–1459.

[36] D. Ancona, A. Ferrando, L. Franceschini, V. Mascardi, Parametric trace expressions for runtime verification of Java-like programs, in: FTfJP@ECOOP, ACM, 2017, pp. 10:1–10:6.

[37] D. Ancona, L. Franceschini, A. Ferrando, V. Mascardi, RML: theory and practice of a domain specific language for runtime verification, Sci. Comput. Program. 205 (2021) 102610. URL: https://doi.org/10.1016/j.scico.2021.102610. doi:10.1016/j.scico.2021.102610.

[38] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (1978) 558–565. URL: http://doi.acm.org/10.1145/359545.359563. doi:10.1145/359545.359563.

[39] D. Briola, V. Mascardi, D. Ancona, Distributed runtime verification of JADE and Jason multiagent systems with Prolog, in: CILC, volume 1195 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014, pp. 319–323.

[40] A. Ferrando, R. C. Cardoso, M. Fisher, D. Ancona, L. Franceschini, V. Mascardi, Rosmonitoring: A runtime verification framework for ROS, in: A. Mohammad, X. Dong, M. Russo (Eds.), Towards Autonomous Robotic Systems - 21st Annual Conference, TAROS 2020, Nottingham, UK, September 16, 2020, Proceedings, volume 12228 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 387–399. URL: https://doi.org/10.1007/978-3-030-63486-5_40. doi:10.1007/978-3-030-63486-5_40.

[41] M. Leotta, D. Ancona, L. Franceschini, D. Olianas, M. Ribaudo, F. Ricca, Towards a runtime verification approach for internet of things systems, in: C. Pautasso, F. Sánchez-Figueroa, K. Systä, J. M. M. Rodriguez (Eds.), Current Trends in Web Engineering - ICWE 2018 International Workshops, MATWEP, EnWot, KD-WEB, WEOD, TourismKG, Cáceres, Spain, June 5, 2018, Revised Selected Papers, volume 11153 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 83–96. URL: https://doi.org/10.1007/978-3-030-03056-8_8. doi:10.1007/978-3-030-03056-8_8.

Forward refutation for Gödel-Dummett Logics

Camillo Fiorentini[1], Mauro Ferrari[2]

[1]Dep. of Computer Science, Università degli Studi di Milano, Italy
[2]Dep. of Theoretical and Applied Sciences, Università degli Studi dell'Insubria, Italy

Abstract
We propose a refutation calculus to check the unprovability of a formula in Gödel-Dummett logics. From refutations we can directly extract countermodels for unprovable formulas, moreover the calculus is designed so to support a forward proof-search strategy that can be understood as a top-down construction of a model.

1. Introduction

With the term Gödel-Dummett logics we refer to the family of intermediate logics GD_k semantically characterised by linear Kripke models of height at most k and the logic GD characterised by linear Kripke models. The logics GD_k were originally introduced by Gödel [1] to study the logics with k-valued matrices semantics, while GD was introduced by Dummett [2] to characterize the logic with infinite valued matrix. Gödel-Dummett logics have been extensively studied for their relations with fuzzy logics [3] and for their computational interpretations [4, 5]. This led to the development of an articulate family of calculi and proof-search strategies for these logics [6, 5, 7, 8, 9, 10].

In this paper we address the problem of defining a logical calculus oriented to generate countermodels for invalid formulas for Gödel-Dummett logics; we exploit the approach based on inverse methods we have developed for Intuitionistic Propositional Logic and the modal logics $\mathbf{K}$ and $\mathbf{S4}$ [11, 12, 8, 13]. The inverse method, introduced by Maslov [14], is a saturation based theorem proving technique closely related to (hyper)resolution [15]; it relies on a forward proof-search strategy and can be applied to cut-free calculi enjoying the subformula property. Given a goal, a set of instances of the rules of the calculus at hand is selected; such specialized rules are repeatedly applied in the forward direction, starting from the axioms (i.e., the rules without premises). Proof-search terminates if either the goal is obtained or the set collecting the proved facts saturates (nothing new can be added). The inverse method has been originally applied to Classical Logic and successively extended to some non-classical logics [16, 15, 17, 18]. In all of the mentioned papers, the inverse method has been exploited to prove the validity of a formula in a specific logic. In [12] we launched a new perspective designing a forward calculus to derive the unprovability of a goal formula in Intuitionistic Propositional Logic and to generate countermodels for unprovable formulas. Differently from other approaches to countermodel

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
fiorentini@di.unimi.it (C. Fiorentini); mauro.ferrari@uninsubria.it (M. Ferrari)
0000-0003-2152-7488 (C. Fiorentini); 0000-0002-7904-1125 (M. Ferrari)

construction for non-classical logics [19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29], where countermodels are obtained as a byproduct of a failed proof-search in a direct or refutation calculus, we define refutation calculi directly supporting model extraction and oriented to forward reasoning. Our approach focuses on countermodel construction; indeed, the rules of the refutation calculus are inspired by the Kripke semantics of the logic at hand and the forward refutation-search procedure can be understood as a top-down method to build a countermodel for the given goal formula. Differently from backward proof-search procedures, forward methods re-use sequents and do not replicate them, accordingly the generated models contain few duplications and are in general very concise.

In this paper we present the refutation calculus for Gödel-Dummett logics, we prove its soundness and completeness and we show how to extract countermodels from its derivations.

2. Preliminaries

Formulas, denoted by uppercase Latin letters, are built from an infinite set of propositional variables $\mathcal{V} = \{p, q, p_1, p_2, \dots\}$, the constant $\bot$ and the connectives $\wedge$, $\vee$, $\supset$; moreover, $\neg A$ stands for $A \supset \bot$. Let G be a formula; $\mathrm{Sf}(G)$ is the set of all subformulas of G (including G itself). By $\textsc{Sl}(G)$ and $\textsc{Sr}(G)$ we denote the subsets of *left* and *right* subformulas of G (a.k.a. negative/positive subformulas of G [30]). Formally, $\textsc{Sl}(G)$ and $\textsc{Sr}(G)$ are the smallest subsets of $\mathrm{Sf}(G)$ such that:

- $G \in \textsc{Sr}(G)$;
- $A \odot B \in \textsc{Sx}(G)$ implies $\{A, B\} \subseteq \textsc{Sx}(G)$, where $\odot \in \{\wedge, \vee\}$ and $\textsc{Sx} \in \{\textsc{Sl}, \textsc{Sr}\}$;
- $A \supset B \in \textsc{Sl}(G)$ implies $B \in \textsc{Sl}(G)$ and $A \in \textsc{Sr}(G)$;
- $A \supset B \in \textsc{Sr}(G)$ implies $B \in \textsc{Sr}(G)$ and $A \in \textsc{Sl}(G)$.

For $\textsc{Sx} \in \{\textsc{Sl}, \textsc{Sr}\}$ we set ($\mathcal{L}^{\supset}$ denotes the set of formulas of the kind $A \supset B$):

$$\begin{array}{llll} \textsc{Sx}^{\mathrm{At}}(G) = \textsc{Sx}(G) \cap \mathcal{V} & & \textsc{Sx}^{\supset}(G) = \textsc{Sx}(G) \cap \mathcal{L}^{\supset} \\ \textsc{Sx}^{\mathrm{At},\supset}(G) = \textsc{Sx}^{\mathrm{At}}(G) \cup \textsc{Sx}^{\supset}(G) & & \mathrm{Sf}^{\mathrm{At}}(G) = \textsc{Sl}^{\mathrm{At}}(G) \cup \textsc{Sr}^{\mathrm{At}}(G) \end{array}$$

A (rooted) Kripke model $\mathcal{K}$ is a quadruple $\langle W, \leq, \rho, V\rangle$ where W is a finite and non-empty set (the set of *worlds*), $\leq$ is a reflexive and transitive binary relation over W, the world ρ (the *root* of $\mathcal{K}$) is the minimum of W w.r.t. $\leq$, and $V : W \mapsto 2^{\mathcal{V}}$ (the *valuation* function) is a map obeying the persistence condition: for every pair of worlds α and β of $\mathcal{K}$, $\alpha \leq \beta$ implies $V(\alpha) \subseteq V(\beta)$; the triple $\langle W, \leq, \rho\rangle$ is called *(Kripke) frame*. We write $\alpha < \beta$ if $\alpha \leq \beta$ and $\alpha \neq \beta$; moreover, we write $\beta \geq \alpha$ ($\beta > \alpha$ resp.) to mean that $\alpha \leq \beta$ ($\alpha < \beta$ resp.). A world β is an *immediate successor* of α in $\mathcal{K}$ if $\alpha < \beta$ and there is no world γ such that $\alpha < \gamma < \beta$.

The valuation V is extended into a *forcing* relation, denoted by $\Vdash$, between worlds of $\mathcal{K}$ and formulas as follows:

$$\begin{array}{ll} \mathcal{K}, \alpha \Vdash p \text{ iff } p \in V(\alpha), \forall p \in \mathcal{V} & \mathcal{K}, \alpha \nVdash \bot \\ \mathcal{K}, \alpha \Vdash A \wedge B \text{ iff } \mathcal{K}, \alpha \Vdash A \text{ and } \mathcal{K}, \alpha \Vdash B & \mathcal{K}, \alpha \Vdash A \vee B \text{ iff } \mathcal{K}, \alpha \Vdash A \text{ or } \mathcal{K}, \alpha \Vdash B \\ \mathcal{K}, \alpha \Vdash A \supset B \text{ iff } \forall \beta \geq \alpha, \mathcal{K}, \beta \Vdash A \text{ implies } \mathcal{K}, \beta \Vdash B. & \end{array}$$

We sometimes write $\alpha \Vdash A$ instead of $\mathcal{K}, \alpha \Vdash A$, leaving understood the model $\mathcal{K}$ at hand when it is clear from the context. By $\alpha \Vdash \Gamma$ we mean that $\alpha \Vdash A$ for every $A \in \Gamma$. A formula A is *valid* in the frame $\langle W, \leq, \rho \rangle$ iff for every valuation V, $\rho \Vdash A$ in the model $\langle W, \leq, \rho, V \rangle$. Propositional Intuitionistic Logic (IPL) is the set of formulas valid in all frames. Accordingly, if there is a model $\mathcal{K}$ such that $\rho \nVdash A$ (here and below ρ designates the root of $\mathcal{K}$), then A is not IPL-valid; we call $\mathcal{K}$ a *countermodel* for A. We write $\Gamma \Vdash A$ iff, for every model $\mathcal{K}$, $\rho \Vdash \Gamma$ implies $\rho \Vdash A$; thus, A is IPL-valid iff $\emptyset \Vdash A$.

Given a frame $\langle W, \leq, \rho \rangle$, the *height* $\mathrm{h}(\alpha)$ of $\alpha \in W$, is defined as follows:

$$\mathrm{h}(\alpha) \;=\; \begin{cases} 0 & \text{if } \alpha \text{ is a maximal world of } W \text{ w.r.t. } \leq \\ 1 + \max\{\, \mathrm{h}(\beta) \mid \alpha < \beta \,\} & \text{otherwise} \end{cases}$$

The *height of* $\mathcal{K}$, denoted by $\mathrm{h}(\mathcal{K})$, is the height of its root.

We say that a *Kripke frame* $\langle W, \leq, \rho \rangle$ *is linear* iff $\leq$ is a linear order over W; i.e., for every pair of worlds α and β, either $\alpha \leq \beta$ or $\beta \leq \alpha$.

Given a formula G we say that a Kripke model $\mathcal{K} = \langle W, \leq, \rho, V \rangle$ is *G-separable* iff, for every pair of worlds α and β in W, the following separation property holds:

- if $\alpha < \beta$, then there is $p \in \mathrm{SL}^{\mathrm{At}}(G) \cap \mathrm{SR}^{\mathrm{At}}(G)$ such that $\mathcal{K}, \alpha \nVdash p$ and $\mathcal{K}, \beta \Vdash p$.

Let Θ be a set of formulas and let us consider the formulas P and N defined by the following grammar, where $A \in \Theta$ and F is any formula

$$\begin{array}{lcl} P & ::= & A \mid P \wedge P \mid F \vee P \mid P \vee F \mid F \supset P \\ N & ::= & A \mid N \vee N \mid F \wedge N \mid N \wedge F \end{array}$$

The *positive closure* of Θ, denoted by $\mathcal{C}l^{+}(\Theta)$, is the smallest set containing the formulas P; the *negative closure* of Θ, denoted by $\mathcal{C}l^{-}(\Theta)$, is the smallest set containing the formulas N. The following properties can be easily proved:

($\mathcal{C}l1$) If $\Theta_1 \subseteq \Theta_2$, then $\mathcal{C}l^{+}(\Theta_1) \subseteq \mathcal{C}l^{+}(\Theta_2)$ and $\mathcal{C}l^{-}(\Theta_1) \subseteq \mathcal{C}l^{-}(\Theta_2)$.

($\mathcal{C}l2$) If $\mathcal{K}, \alpha \Vdash A$, for every $A \in \Theta$, and $P \in \mathcal{C}l^{+}(\Theta)$, then $\mathcal{K}, \alpha \Vdash P$.

($\mathcal{C}l3$) If $\mathcal{K}, \alpha \nVdash A$, for every $A \in \Theta$, and $N \in \mathcal{C}l^{-}(\Theta)$, then $\mathcal{K}, \alpha \nVdash N$.

The logics GD_k and GD

In this paper we consider the Gödel-Dummett logics GD_k ($k \geq 0$) and GD defined as follows (see [31]):

- GD_k is the set of formulas valid in linear models $\mathcal{K}$ such that $\mathrm{h}(\mathcal{K}) \leq k$;
- $\mathrm{GD} = \bigcap_{k \geq 0} \mathrm{GD}_k$.

We remark that $\mathrm{IPL} \subset \mathrm{GD} \subset \cdots \subset \mathrm{GD}_2 \subset \mathrm{GD}_1 \subset \mathrm{GD}_0 = \mathrm{CPL}$, where CPL is the set of classically valid formulas.

3. The GD-refutation calculus

The forward refutation calculus $\mathbf{R}_{\mathbf{GD}}(G)$ is a calculus to infer the unprovability of a formula G (the *goal formula*) in GD_k and it is designed to support forward refutation-search (for a presentation of forward calculi we refer to [15]). The calculus acts on $\mathbf{R}_{\mathbf{GD}}(G)$-*sequents*[1] having the form $\Gamma \not\Rightarrow_k \Lambda\,;\,\Delta$ where:

- $k \geq 0$, $\Gamma \subseteq \mathrm{S}\textsc{l}^{\mathrm{At},\supset}(G)$, $\Lambda \subseteq \mathrm{S}\textsc{l}^{\mathrm{At}}(G) \cap \mathrm{S}\textsc{r}^{\mathrm{At}}(G)$, and $\Delta \subseteq \mathrm{S}\textsc{r}^{\mathrm{At},\supset}(G)$;
- if $k = 0$, then $\Lambda = \emptyset$.

The *rank* of $\sigma = \Gamma \not\Rightarrow_k \Lambda\,;\,\Delta$, denoted by $\mathrm{Rn}(\sigma)$, is k. We will see that, whenever there exists a refutation $\mathcal{D}$ of σ in the calculus $\mathbf{R}_{\mathbf{GD}}(G)$, from $\mathcal{D}$ we can extract a model containing a world α such that $\mathrm{h}(\alpha) = k$ and:

- $\mathcal{K}, \alpha \Vdash \bigwedge \Gamma$ and $\mathcal{K}, \alpha \nVdash \bigvee \Delta$; moreover, for every $A \supset B \in \Gamma$, $\mathcal{K}, \alpha \nVdash A$;
- if $k > 0$, then Λ is the set of propositional variables forced in the immediate successor of α and not in α.

The rules of $\mathbf{R}_{\mathbf{GD}}(G)$ are displayed in Fig. 1. We point out that the formulas introduced in the conclusion of the rules in the left (side of the sequents) must belong to $\mathrm{S}\textsc{l}(G)$ and the formulas introduced in the right must belong to $\mathrm{S}\textsc{r}(G)$. An $\mathbf{R}_{\mathbf{GD}}(G)$-sequent σ is *saturated* if none of the rules $L \supset$ and $R \supset$ can be applied to σ. As a consequence of the side condition, the application of the rule Succ is delayed until a saturated sequent is get. The successor rule Succ moves the propositional variables in Λ' from the left side of sequent to the right side; in countermodel construction, an application of the Succ rule corresponds to a downward expansion of a model, obtained by adding a new root ρ' below the current root ρ; the propositional variables in Λ' are forced in ρ and not in ρ'. Note that, given a rule and the sequent in the premise, we can build different instances of the rule according with the non-deterministic choices described in the side-condition of the rule. E.g., we can generate a different instance of $L \supset$ having $\Gamma \not\Rightarrow_0 \cdot\,;\,\Delta$ in the premise, for every $A \supset B \in \mathrm{S}\textsc{l}(G)$ such that $A \supset B \notin \Gamma$ and $A \in \mathcal{C}l^{-}(\Delta)$. This also holds for the axiom-rule; we get a different axiom for every possible partition $(\Gamma^{\mathrm{At}}, \Delta^{\mathrm{At}})$ of $\mathrm{Sf}^{\mathrm{At}}(G)$. A *proof tree* of the calculus $\mathbf{R}_{\mathbf{GD}}(G)$ is a tree having $\mathbf{R}_{\mathbf{GD}}(G)$-sequents as nodes and built according to the rules of $\mathbf{R}_{\mathbf{GD}}(G)$ (see e.g. [30] for a formal definition). Note that all the proof trees of $\mathbf{R}_{\mathbf{GD}}(G)$ are linear. We introduce some definitions:

- $\mathcal{D}$ is an $\mathbf{R}_{\mathbf{GD}}(G)$-*refutation of* σ iff $\mathcal{D}$ is a proof tree of $\mathbf{R}_{\mathbf{GD}}(G)$ having σ as root sequent; the rank of $\mathcal{D}$ is the rank of σ ($\mathrm{Rn}(\mathcal{D}) = \mathrm{Rn}(\sigma)$).
- $\mathcal{D}$ is an $\mathbf{R}_{\mathbf{GD}}(G)$-*refutation of* G iff $\mathcal{D}$ is an $\mathbf{R}_{\mathbf{GD}}(G)$-refutation of $\Gamma \not\Rightarrow_k \Lambda\,;\,\Delta$ and $G \in \mathcal{C}l^{-}(\Delta \cup \Lambda)$.
- $\vdash_G^k G$ iff there is an $\mathbf{FRJ}(G)$-refutation $\mathcal{D}$ of G such that $\mathrm{Rn}(\mathcal{D}) \leq k$.

[1]In refutation calculi sequents are sometimes called *anti-sequents* (see,e.g., [27]).

$$\frac{}{\Gamma^{\mathrm{At}} \not\Rightarrow_0 \cdot\,;\, \Delta^{\mathrm{At}}, \bot}\ \mathrm{Ax} \qquad \begin{array}{l}\Gamma^{\mathrm{At}} \cup \Delta^{\mathrm{At}} = \mathrm{Sf}^{\mathrm{At}}(G)\\ \Gamma^{\mathrm{At}} \cap \Delta^{\mathrm{At}} = \emptyset\end{array}$$

$$\frac{\Gamma \not\Rightarrow_0 \cdot\,;\, \Delta}{A \supset B, \Gamma \not\Rightarrow_0 \cdot\,;\, \Delta}\ L\supset \quad \begin{array}{l}A \supset B \notin \Gamma \cup \Delta\\ A \in \mathcal{C}l^-(\Delta)\end{array} \qquad \frac{\Gamma \not\Rightarrow_k \Lambda\,;\, \Delta}{A \supset B, \Gamma \not\Rightarrow_k \Lambda\,;\, \Delta}\ L\supset \quad \begin{array}{l}A \supset B \notin \Gamma \cup \Delta\\ A \in \mathcal{C}l^-(\Delta \cup \Lambda)\\ B \in \mathcal{C}l^+(\Gamma \cup \Lambda)\end{array}$$

$$\frac{\Gamma \not\Rightarrow_k \Lambda\,;\, \Delta}{\Gamma \not\Rightarrow_k \Lambda\,;\, \Delta, A \supset B}\ R\supset \quad \begin{array}{l}A \supset B \notin \Delta \cup \Delta\\ A \in \mathcal{C}l^+(\Gamma)\\ B \in \mathcal{C}l^-(\Delta \cup \Lambda)\end{array}$$

$$\frac{\Gamma \not\Rightarrow_k \Lambda\,;\, \Delta}{\Gamma \setminus \Lambda' \not\Rightarrow_{k+1} \Lambda'\,;\, \Delta, \Lambda}\ \mathrm{Succ} \quad \begin{array}{l}\Gamma \not\Rightarrow_k \Lambda\,;\, \Delta \text{ is saturated}\\ \emptyset \subset \Lambda' \subseteq \Gamma \cap \mathcal{V}\end{array}$$

Figure 1: The refutation calculus $\mathbf{R}\textsc{gd}(G)$.

Example 1 Let us consider the following formula G:

$$G \;=\; A \vee (p \supset r) \vee B \vee (C \supset (p \vee \neg p))$$
$$A \;=\; \neg(q \wedge r) \qquad B \;=\; (\neg\neg p \wedge (p \supset q)) \supset q \qquad C \;=\; B \supset (\neg\neg p \wedge q)$$

We search for an $\mathbf{R}\textsc{gd}(G)$-derivation building a database of proved sequents according with the naive recipe of [15]: we start by inserting all the axioms; then we enter a loop where, at each iteration, we apply all the possible rules to the sequents collected in previous steps. The loop ends if either a sequent $\Gamma \not\Rightarrow_k \Lambda\,;\, \Delta$ with $G \in \mathcal{C}l^-(\Delta \cup \Lambda)$ is generated or no new sequent can be added to the database (the database is saturated). Fig. 2 shows the fragment of the database containing the sequents needed to get the $\mathbf{R}\textsc{gd}(G)$-derivation of G. In the example, we denote with $\sigma_{(j)}$ the sequent at line (j) of Fig. 2. As an example, the sequent $\sigma_{(2)}$ is obtained by applying the rule $R \supset$ to the sequent $\sigma_{(1)}$, i.e,:

$$\frac{p,\ q,\ r \not\Rightarrow_0 \cdot\,;\, \bot}{p,\ q,\ r \not\Rightarrow_0 \cdot\,;\, \bot,\ \neg p}\ R\supset$$

recalling that $\neg p = p \supset \bot$; note that, $p \in \mathcal{C}l^+(\{p,\ q,\ r\})$ and $\bot \in \mathcal{C}l^-(\{\bot\})$. As for sequent $\sigma_{(3)}$ it is obtained by applying the $L \supset$ rule to $\sigma_{(2)}$:

$$\frac{p,\ q,\ r \not\Rightarrow_0 \cdot\,;\, \bot,\ \neg p}{p,\ q,\ r \not\Rightarrow_0 \cdot\,;\, \bot,\ \neg p,\ A}\ L\supset$$

where $A = \neg(q \wedge r) = (q \wedge r) \supset \bot$, note that $\bot \in \mathcal{C}l^-(\{\bot\})$ and $q \wedge r \in \mathcal{C}l^+(\{p,\ q,\ r\})$. Sequent $\sigma_{(5)}$ is obtained applying Succ to $\sigma_{(4)}$ by moving r from left to right; similarly, $\sigma_{(7)}$

$$
\begin{array}{l}
G \;=\; A \lor (p \supset r) \lor B \lor (C \supset (p \lor \neg p)) \\
A \;=\; \neg(q \land r) \qquad B \;=\; (\neg\neg p \land (p \supset q)) \supset q \qquad C \;=\; B \supset (\neg\neg p \land q) \\
\textsc{Sl}^{\mathrm{At}}(G) \;=\; \{\, p,\ q,\ r \,\} \qquad \textsc{Sl}^{\supset}(G) \;=\; \{\, C,\ \neg\neg p,\ p \supset q \,\} \\
\textsc{Sr}^{\mathrm{At}}(G) \;=\; \{\, p,\ q,\ r \,\} \qquad \textsc{Sr}^{\supset}(G) \;=\; \{\, A,\ p \supset r,\ B,\ C \supset (p \lor \neg p),\ \neg p \,\}
\end{array}
$$

$$
\begin{array}{lrcll}
(1) & p,\ q,\ r & \not\Rightarrow_0 & \cdot\,;\ \bot & \mathrm{Ax} \\
(2) & p,\ q,\ r & \not\Rightarrow_0 & \cdot\,;\ \bot,\ \neg p & R\supset\ (1) \\
(3) & p,\ q,\ r & \not\Rightarrow_0 & \cdot\,;\ \bot,\ \neg p,\ A & R\supset\ (2) \\
(4) & \neg\neg p,\ p,\ q,\ r & \not\Rightarrow_0 & \cdot\,;\ \bot,\ \neg p,\ A \quad (*) & L\supset\ (3) \\
\hline
(5) & \neg\neg p,\ p,\ q & \not\Rightarrow_1 & r\,;\ \bot,\ \neg p,\ A & \mathrm{Succ}\ (4) \\
(6) & \neg\neg p,\ p,\ q & \not\Rightarrow_1 & r\,;\ \bot,\ \neg p,\ A,\ p \supset r \quad (*) & R\supset\ (5) \\
\hline
(7) & \neg\neg p & \not\Rightarrow_2 & p,\ q\,;\ \bot,\ \neg p,\ A, p \supset r,\ r & \mathrm{Succ}\ (6) \\
(8) & p \supset q,\ \neg\neg p & \not\Rightarrow_2 & p,\ q\,;\ \bot,\ \neg p,\ A, p \supset r,\ r & L\supset\ (7) \\
(9) & p \supset q,\ \neg\neg p & \not\Rightarrow_2 & p,\ q\,;\ \bot,\ \neg p,\ A, p \supset r,\ r,\ B & R\supset\ (8) \\
(10) & C,\ p \supset q,\ \neg\neg p & \not\Rightarrow_2 & p,\ q\,;\ \bot,\ \neg p,\ A, p \supset r,\ r,\ B & L\supset\ (9) \\
(11) & C,\ p \supset q,\ \neg\neg p & \not\Rightarrow_2 & p,\ q\,;\ \bot,\ \neg p,\ A, p \supset r,\ r,\ B,\ C \supset (p \lor \neg p) \quad (*) & R\supset\ (10)
\end{array}
$$

Figure 2: Building the $\mathbf{R}\textsc{gd}(G)$-refutation of G; p-sequents are marked by (*).

is obtained applying Succ to $\sigma_{(6)}$ by moving p and q from left to right and moving r to the rightmost zone. We have marked with $*$ the premises of Succ that, as we discuss later, play a role in the construction of the countermodel. Note that sequent $\sigma_{(11)}$ meets the property $G \in \mathcal{C}l^{-}(\Delta \cup \Lambda)$. The tree-like structure of the $\mathbf{R}\textsc{gd}(G)$*-refutation of* G is displayed in the left of Fig. 3. $\Diamond$

We introduce the following relations between $\mathbf{R}\textsc{gd}(G)$-sequents:

- $\sigma_1 \overset{\mathcal{R}}{\mapsto}_0 \sigma_2$ iff $\mathcal{R}$ is a rule of $\mathbf{R}\textsc{gd}(G)$ having premise σ_1 and conclusion σ_2;
- $\sigma_1 \mapsto_0 \sigma_2$ iff there exists a rule $\mathcal{R}$ such that $\sigma_1 \overset{\mathcal{R}}{\mapsto}_0 \sigma_2$;
- $\sigma_1 \bar{\mapsto}_0 \sigma_2$ iff there exists a rule $\mathcal{R} \neq$ Succ such that $\sigma_1 \overset{\mathcal{R}}{\mapsto}_0 \sigma_2$;
- $\mapsto_*$ (resp. $\bar{\mapsto}_*$) is the reflexive and transitive closure of $\mapsto$ (resp. $\bar{\mapsto}_*$).

The following properties can be easily proved ($||\Theta||$ denotes the cardinality of the set Θ)

Lemma 1 Let $\sigma_1 = \Gamma_1 \not\Rightarrow_{k_1} \Lambda_1 \,;\, \Delta_1$ and $\sigma_2 = \Gamma_2 \not\Rightarrow_{k_2} \Lambda_2 \,;\, \Delta_2$ be two $\mathbf{R}_{\mathbf{GD}}(G)$-sequents such that $\sigma_1 \mapsto_* \sigma_2$. Then:

(i) $k_1 \leq k_2$.

(ii) $\Gamma_1 \cap \mathcal{L}^{\supset} \subseteq \Gamma_2 \cap \mathcal{L}^{\supset}$ and $\Gamma_2 \cap \mathcal{V} \subseteq \Gamma_1$. Moreover, if $k_1 = k_2$ then $\Gamma_1 \subseteq \Gamma_2$ and $\Gamma_2 \cap \mathcal{V} = \Gamma_1 \cap \mathcal{V}$.

(iii) If $k_1 = k_2$, then $\Lambda_1 = \Lambda_2$ and $\Delta_1 \subseteq \Delta_2$. If $k_1 < k_2$, then $\Delta_1 \cup \Lambda_1 \subseteq \Delta_2$ and $\Lambda_2 \subseteq \Gamma_1$.

(iv) $k_2 \leq k_1 + ||\Gamma_1 \cap \mathcal{V}||$.

By Lemma 1, we get:

Proposition 1 The relation $\mapsto_0$ on $\mathbf{R}_{\mathbf{GD}}(G)$-sequents is terminating.

Proof. Each application of rules $L \supset$ and $R \supset$ introduces a new subformula of G in the conclusion, thus $\bar{\mapsto}_0$ is terminating, Accordingly, an infinite $\mapsto_0$-chain starting from $\Gamma \not\Rightarrow_k \Lambda \,;\, \Delta$ should contain infinitely many applications of rule Succ. This is not possible, since every application of rule Succ increases by 1 the rank of a sequent and, by Lemma 1(iv), the rank of any sequent in the chain is bounded by $k + ||\Gamma \cap \mathcal{V}||$. We conclude that $\mapsto_0$ is terminating. □

4. Soundness

Soundness of $\mathbf{R}_{\mathbf{GD}}(G)$ is stated as follows:

Theorem 1 (Soundness of $\mathbf{R}_{\mathbf{GD}}(G)$) $\vdash_G^k G$ implies $G \notin \mathrm{GD}_k$.

To prove this, we show that from an $\mathbf{R}_{\mathbf{GD}}(G)$-refutation $\mathcal{D}$ of G we can extract a countermodel $\mathrm{Mod}(\mathcal{D})$ for G such that $\mathrm{h}(\mathrm{Mod}(\mathcal{D})) = \mathrm{Rn}(\mathcal{D})$.

Let $\mathcal{D}$ an $\mathbf{R}_{\mathbf{GD}}(G)$-refutation and let σ be a sequent occurring in $\mathcal{D}$; σ is a *p-sequent* (*prime sequent*) iff σ is saturated or σ is the root sequent of $\mathcal{D}$. Let $\mathrm{Mod}(\mathcal{D}) = \langle\, \mathrm{P}(\mathcal{D}),\, \leq,\, \rho, V\, \rangle$ where:

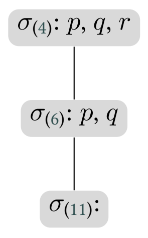
- $\mathrm{P}(\mathcal{D})$ is the set of all p-sequents occurring in $\mathcal{D}$;
- for every $\sigma_1, \sigma_2 \in \mathrm{P}(\mathcal{D})$, $\sigma_1 \leq \sigma_2$ iff $\sigma_2 \mapsto_* \sigma_1$;
- ρ is the root of $\mathcal{D}$;
- V maps a p-sequent $\Gamma \not\Rightarrow_k \Lambda \,;\, \Delta$ to the set $\Gamma \cap \mathcal{V}$.

Then, since $\mathbf{R}_{\mathbf{GD}}(G)$-refutations are linear, $\mathrm{Mod}(\mathcal{D})$ is a linear model; note that, by Lemma1(ii), $\sigma_1 \leq \sigma_2$ implies $V(\sigma_1) \subseteq V(\sigma_2)$, hence the definition of V is sound. We call $\mathrm{Mod}(\mathcal{D})$ the *model extracted from* $\mathcal{D}$. For every sequent σ occurring in $\mathcal{D}$, let $\phi(\sigma)$ be the p-sequent in $\mathcal{D}$ immediately below σ, namely:

$$\phi(\sigma) \;=\; \sigma_p \qquad \text{iff} \quad \sigma_p \in \mathrm{P}(\mathcal{D}) \text{ and } \sigma \,\bar{\mapsto}_*\, \sigma_p$$

It is easy to check that:

$$\begin{array}{ll}
\overline{\phantom{\sigma_{(10)}}} & \text{Ax} \\
\sigma_{(1)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & R\supset \\
\sigma_{(2)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & R\supset \\
\sigma_{(3)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & L\supset \\
\sigma^{*}_{(4)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & \text{Succ} \\
\sigma_{(5)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & R\supset \\
\sigma^{*}_{(6)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & \text{Succ} \\
\sigma_{(7)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & L\supset \\
\sigma_{(8)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & R\supset \\
\sigma_{(9)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & L\supset \\
\sigma_{(10)} & \\[-4pt]
\overline{\phantom{\sigma_{(10)}}} & R\supset \\
\sigma^{*}_{(11)} &
\end{array}$$



sequents labeled by $(*)$ are p-sequents
$\sigma_{(j)}$ refers to the sequent at line (j)

$$\phi(\sigma_j) = \begin{cases} \sigma_{(4)} & j = 1,2,3,4 \\ \sigma_{(6)} & j = 5,6 \\ \sigma_{(11)} & j = 7,8,9,10,11 \end{cases}$$

Figure 3: The $\mathbf{R}_{\mathrm{GD}}(G)$-derivation of G and the extracted countermodel.

- p-sequents are fixed points of ϕ, i.e., $\sigma_p \in \mathrm{P}(\mathcal{D})$ implies $\phi(\sigma_p) = \sigma_p$;
- ϕ is a surjective map from the set of sequents of $\mathcal{D}$ onto $\mathrm{P}(\mathcal{D})$;
- $\sigma_1 \mapsto_* \sigma_2$ implies $\phi(\sigma_2) \leq \phi(\sigma_1)$;
- $\mathrm{h}(\phi(\sigma)) = \mathrm{Rn}(\sigma)$.

We call ϕ the *map associated with* $\mathcal{D}$; note that $\mathrm{Mod}(\mathcal{D})$ is G-separable.

Example 2 The model $\mathrm{Mod}(\mathcal{D}_G)$ and the related map ϕ are shown in Fig. 3. The bottom world is the root and $\sigma < \sigma'$ iff the world σ is drawn below σ'. For each σ, we display the set $V(\sigma)$. As an example, $V(\sigma_4) = \{p, q, r\}$. It is easy to check that $\sigma_{(11)} \nVdash G$. $\Diamond$

The following lemma is the main step to prove the soundness theorem:

Lemma 2 Let $\mathcal{D}$ be an $\mathbf{R}_{\mathrm{GD}}(G)$-refutation, let $\mathrm{Mod}(\mathcal{D})$ be the model extracted from $\mathcal{D}$ and ϕ the map associated with $\mathcal{D}$. For every sequent $\sigma = \Gamma \not\Rightarrow_k \Lambda\,;\,\Delta$ occurring in $\mathcal{D}$, the following properties hold.

(i) For every $C \in \Gamma$, $\mathrm{Mod}(\mathcal{D}), \phi(\sigma) \Vdash C$. Moreover, if $C = A \supset B$, then $\mathrm{Mod}(\mathcal{D}), \phi(\sigma) \nVdash A$.

(ii) For every $C \in \Delta \cup \Lambda$, $\mathrm{Mod}(\mathcal{D}), \phi(\sigma) \nVdash C$.

Proof. By induction on the height of σ in $\mathcal{D}$, taking into account the closure properties $(\mathcal{C}l1)$-$(\mathcal{C}l2)$ and Lemma 1. $\square$

Let $\vdash^k_G G$. Then, there exists an $\mathbf{R}_{\mathrm{GD}}(G)$-refutation $\mathcal{D}$ of $\sigma = \Gamma \not\Rightarrow_{k'} \Lambda\,;\,\Delta$ such that $k' \leq k$ and $G \in \mathcal{C}l^-(\Delta \cup \Lambda)$. Let $\mathrm{Mod}(\mathcal{D}) = \langle P, \leq, \rho, V\rangle$ and ϕ the associated map. We have $\mathrm{h}(\mathrm{Mod}(\mathcal{D})) = k' \leq k$ and $\phi(\sigma) = \rho$; by Lemma 2(ii), we get $\mathrm{Mod}(\mathcal{D}), \rho \nVdash C$, for every $C \in \Delta \cup \Lambda$. Since $G \in \mathcal{C}l^-(\Delta \cup \Lambda)$, by property $(\mathcal{C}l3)$ of negative closures $\mathrm{Mod}(\mathcal{D}), \rho \nVdash G$, hence $G \notin \mathrm{GD}_k$. This proves the soundness of $\mathbf{R}_{\mathrm{GD}}(G)$ (Theorem 1).

5. Completeness

We prove the completeness of $\mathbf{R}_{\mathrm{GD}}(G)$:

Theorem 2 (Completeness of $\mathbf{R}_{\mathrm{GD}}(G)$) $G \notin \mathrm{GD}_k$ implies $\vdash^k_G G$.

The proof goes along the following lines. First we show that we can use a G-separable countermodel of G of height k to build an $\mathbf{R}_{\mathrm{GD}}(G)$-refutation of G with rank k at most. Then, we prove that, given a formula $G \notin \mathrm{GD}_k$, there exists a G-separable model $\mathcal{K} = \langle K, \leq, \rho, V\rangle$ of height at most k such that $\mathcal{K}, \rho \nVdash G$.

The following properties of saturated sequents can be easily proved.

Lemma 3 Let $\sigma = \Gamma \not\Rightarrow_k \Lambda\,;\, \Delta$ be a saturated $\mathbf{R}_{\mathrm{GD}}(G)$-sequent. Then:

(i) If $k = 0$ and $A \supset B \in \mathrm{S}_{\mathrm{L}}(G)$ and $A \in \mathcal{C}l^-(\Delta)$, then $A \supset B \in \Gamma$.

(ii) If $A \supset B \in \mathrm{S}_{\mathrm{L}}(G)$ and $A \in \mathcal{C}l^-(\Delta \cup \Lambda)$ and $B \in \mathcal{C}l^+(\Gamma \cup \Lambda)$, then $A \supset B \in \Gamma$.

(iii) If $A \supset B \in \mathrm{S}_{\mathrm{R}}(G)$ and $A \in \mathcal{C}l^+(\Gamma)$ and $B \in \mathcal{C}l^-(\Delta \cup \Lambda)$, then $A \supset B \in \Delta$.

Lemma 4 For every $\mathbf{R}_{\mathrm{GD}}(G)$-sequent σ, there exists a unique saturated $\mathbf{R}_{\mathrm{GD}}(G)$-sequent σ' such that $\sigma \mapsto^{\!\!\!\!-}_* \sigma'$.

Proof. Let $\mathcal{S}_G$ be the set of all the $\mathbf{R}_{\mathrm{GD}}(G)$-sequents and let us consider the Abstract Reduction System $\mathcal{A}_G = \langle \mathcal{S}_G, \bar{\mapsto}\rangle$ (see e.g. [32]). By Proposition 1, $\mathcal{A}_G$ is terminating; the irreducible elements of $\mathcal{A}_G$ are the saturated sequents. Moreover, one can easily check that $\mathcal{A}_G$ is locally confluent; indeed, if $\sigma \bar{\mapsto} \sigma_1$ and $\sigma \bar{\mapsto} \sigma_2$, there exists σ' such that $\sigma_1 \bar{\mapsto} \sigma'$ and $\sigma_2 \bar{\mapsto} \sigma'$. By Newman's Lemma [32], $\mathcal{A}_G$ is confluent, and this proves the assertion. □

Let σ be an $\mathbf{R}_{\mathrm{GD}}(G)$-sequent; by σ^* we denote the unique saturated $\mathbf{R}_{\mathrm{GD}}(G)$-sequent such that $\sigma \bar{\mapsto}_* \sigma^*$ (thus, if σ is saturated, we have $\sigma^* = \sigma$).

Let $\mathcal{K} = \langle W, \leq, \rho, V\rangle$ be a G-separable model. For every $\alpha \in W$, we define the saturated $\mathbf{R}_{\mathrm{GD}}(G)$-sequent $\mathrm{Sat}_G(\alpha)$ associated with α by induction on $\mathrm{h}(\alpha)$.

- $\mathrm{h}(\alpha) = 0$.

$$\mathrm{Sat}_G(\alpha) \;=\; (\,\Gamma^{\mathrm{At}} \not\Rightarrow_0 \cdot\,;\, \Delta^{\mathrm{At}} \perp)^* \qquad \begin{aligned}\Gamma^{\mathrm{At}} &= \{\, p \in \mathrm{S}_{\mathrm{L}}^{\mathrm{At}}(G) \mid \mathcal{K}, \alpha \Vdash p\,\}\\ \Delta^{\mathrm{At}} &= \{\, p \in \mathrm{S}_{\mathrm{R}}^{\mathrm{At}}(G) \mid \mathcal{K}, \alpha \nVdash p\,\}\end{aligned}$$

- $\mathrm{h}(\alpha) > 0$.

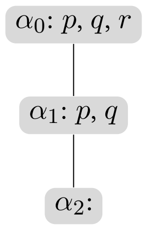
Let β be the immediate successor of α, let $\mathrm{Sat}_G(\beta) = \Gamma \not\Rightarrow_k \Lambda\,;\,\Delta$ and let

$$\Lambda_\beta \;=\; \{\, p \in \mathrm{S}_{\mathrm{L}}^{\mathrm{At}}(G) \cap \mathrm{S}_{\mathrm{R}}^{\mathrm{At}}(G) \mid \mathcal{K}, \beta \Vdash p \text{ and } \mathcal{K}, \alpha \nVdash p\,\}$$

Note that Λ_β is not empty (indeed, $\mathcal{K}$ is G-separable). We set:

$$\mathrm{Sat}_G(\alpha) \;=\; (\,\Gamma \setminus \Lambda_\beta \not\Rightarrow_{k+1} \Lambda_\beta\,;\, \Delta, \Lambda\,)^*$$

Example 3 Let G be defined as in Ex. 1. Below we display a G-separable model $\mathcal{K}$ consisting of three worlds α_0, α_1, α_2; for each α_j, the saturated set $\mathrm{Sat}_G(\alpha_j)$ coincides with one of the saturated sequents occurring in the refutation in Fig. 2.



$$\mathrm{Sat}_G(\alpha_0) = \neg\neg p,\ p,\ q,\ r \not\Rightarrow_0 \cdot\,;\ \bot,\ \neg p,\ A \qquad (\sigma_{(4)})$$
$$\mathrm{Sat}_G(\alpha_1) = \neg\neg p,\ p,\ q \not\Rightarrow_1 r\,;\ \bot,\ \neg p,\ A,\ p \supset r \qquad (\sigma_{(6)})$$
$$\mathrm{Sat}_G(\alpha_2) = C,\ p \supset q,\ \neg\neg p \not\Rightarrow_2 p,\ q\,;\ \bot,\ \neg p,\ A, p \supset r,\ r,\ B,\ C \supset (p \vee \neg p) \qquad (\sigma_{(11)})$$

◊

Lemma 5 Let $\mathcal{K} = \langle W, \leq, \rho, V\rangle$ be a G-separable model, let $\alpha \in W$ and $\mathrm{Sat}_G(\alpha) = \Gamma \not\Rightarrow_k \Lambda\,;\,\Delta$. Then:

(i) $k = \mathrm{h}(\alpha)$.

(ii) If $p \in \mathrm{S}\textsc{l}^{\mathrm{At}}(G)$ and $\mathcal{K}, \alpha \Vdash p$, then $p \in \Gamma$.

(iii) If $p \in \mathrm{S}\textsc{r}^{\mathrm{At}}(G)$ and $\mathcal{K}, \alpha \nVdash p$, then $p \in \Delta \cup \Lambda$.

(iv) There exists an $\mathbf{R}\textsc{gd}(G)$-refutation of $\mathrm{Sat}_G(\alpha)$.

(v) If $C \in \mathrm{S}\textsc{l}(G)$ and $\mathcal{K}, \alpha \Vdash C$, then $C \in \mathcal{C}l^+(\Gamma)$.

(vi) If $C \in \mathrm{S}\textsc{r}(G)$ and $\mathcal{K}, \alpha \nVdash C$, then $C \in \mathcal{C}l^-(\Delta \cup \Lambda)$.

Proof. Points (i)-(iv) easily follow by induction on $\mathrm{h}(\alpha)$. We prove (v) and (vi) by a main induction hypothesis (IH1) on $\mathrm{h}(\alpha)$ and a secondary induction hypothesis (IH2) on $|C|$. Note that, by point (i), we have $k = \mathrm{h}(\alpha)$.

(C1) $\mathrm{h}(\alpha) = 0$.

We have $k = 0$, hence $\Lambda = \emptyset$. Let $C \in \mathrm{S}\textsc{l}(G)$ such that $\mathcal{K}, \alpha \Vdash C$; we show $C \in \mathcal{C}l^+(\Gamma)$. If $C \in \mathcal{V}$, by point (ii) we get $p \in \Gamma$, hence $p \in \mathcal{C}l^+(\Gamma)$. Let $C = A \wedge B$. Then, $\mathcal{K}, \alpha \Vdash A$ and $\mathcal{K}, \alpha \Vdash B$. By (IH2), we get $A \in \mathcal{C}l^+(\Gamma)$ and $B \in \mathcal{C}l^+(\Gamma)$, hence $A \wedge B \in \mathcal{C}l^+(\Gamma)$. The case $C = A \vee B$ is similar. Let $C = A \supset B$. If $\mathcal{K}, \alpha \Vdash B$, by (IH2) we get $B \in \mathcal{C}l^+(\Gamma)$, hence $A \supset B \in \mathcal{C}l^+(\Gamma)$. Let us assume $\mathcal{K}, \alpha \nVdash B$. Then $\mathcal{K}, \alpha \nVdash A$ hence, by (IH2), we get $A \in \mathcal{C}l^-(\Delta)$. By point Lemma 3(i) it follows that $A \supset B \in \Gamma$, hence $A \supset B \in \mathcal{C}l^+(\Gamma)$. This concludes the proof of (v).

Let $C \in \textsc{Sr}(G)$ such that $\mathcal{K}, \alpha \nVdash C$; we show $C \in \mathcal{C}l^-(\Delta)$. If $C \in \mathcal{V}$, by point (iii) we get $C \in \Delta$, hence $C \in \mathcal{C}l^-(\Delta)$. Let $C = A \wedge B$. Then, $\mathcal{K}, \alpha \nVdash A$ or $\mathcal{K}, \alpha \nVdash B$. According to the case, by (IH2) we get $A \in \mathcal{C}l^-(\Delta)$ or $B \in \mathcal{C}l^-(\Delta)$, hence $A \wedge B \in \mathcal{C}l^-(\Delta)$. The case $C = A \vee B$ is similar. Let $C = A \supset B$. We have $\mathcal{K}, \alpha \Vdash A$ and $\mathcal{K}, \alpha \nVdash B$. By (IH2), we get $A \in \mathcal{C}l^+(\Gamma)$ and $B \in \mathcal{C}l^-(\Delta)$. By Lemma 3(iii) it follows that $A \supset B \in \Delta$, hence $A \supset B \in \mathcal{C}l^+(\Delta)$. This concludes the proof of (vi).

(C2) $\mathrm{h}(\alpha) > 0$.

Let β be the immediate successor of α (thus, $\mathrm{h}(\beta) = \mathrm{h}(\alpha) - 1$) and let:

$$\begin{array}{rcl} \mathrm{Sat}_G(\beta) & = & \Gamma' \not\Rightarrow_{k-1} \Lambda' \,;\, \Delta' \\ \Lambda_\beta & = & \{\, p \in \textsc{Sl}^{\mathrm{At}}(G) \cap \textsc{Sl}^{\mathrm{At}}(G) \mid \mathcal{K}, \beta \Vdash p \text{ and } \mathcal{K}, \alpha \nVdash p \,\} \end{array}$$

We have:

$$\mathrm{Sat}_G(\alpha) \;=\; (\,\Gamma' \setminus \Lambda_\beta \not\Rightarrow_k \Lambda_\beta \,;\, \Delta', \Lambda'\,)^* \qquad \Gamma' \setminus \Lambda_\beta \subseteq \Gamma \qquad \Delta' \cup \Lambda' \subseteq \Delta$$

Let $C \in \textsc{Sl}(G)$ such that $\mathcal{K}, \alpha \Vdash C$; we show $C \in \mathcal{C}l^+(\Gamma)$. The cases $C \in \mathcal{V}$, $C = A \wedge B$ and $C = A \vee B$ can be proved as in the case (C1). Let $C = A \supset B$. If $\mathcal{K}, \alpha \Vdash B$ then, by (IH2), $B \in \mathcal{C}l^+(\Gamma)$, which implies $A \supset B \in \mathcal{C}l^+(\Gamma)$. Let us assume $\mathcal{K}, \alpha \nVdash B$; we show that $A \supset B \in \Gamma$. Since $\alpha < \beta$, it holds that $\mathcal{K}, \beta \Vdash A \supset B$. By (IH1), $A \supset B \in \mathcal{C}l^+(\Gamma')$, hence $B \in \mathcal{C}l^+(\Gamma')$ or $A \supset B \in \Gamma'$. In the latter case, since $A \supset B \in \Gamma' \setminus \Lambda_\beta$ and $\Gamma' \setminus \Lambda_\beta \subseteq \Gamma$, we get $A \supset B \in \Gamma$. Let us consider the former case (namely, $B \in \mathcal{C}l^+(\Gamma')$). From $\Gamma' \setminus \Lambda_\beta \subseteq \Gamma$, it follows that $\Gamma' \subseteq \Gamma \cup \Lambda_\beta$, hence $B \in \mathcal{C}l^+(\Gamma \cup \Lambda_\beta)$. Since $\mathcal{K}, \alpha \Vdash A \supset B$ and $\mathcal{K}, \alpha \nVdash B$, it holds that $\mathcal{K}, \alpha \nVdash A$ hence, by (IH2), $A \in \mathcal{C}l^-(\Delta \cup \Lambda)$. We can apply Lemma 3(ii), and infer that $A \supset B \in \Gamma$. Having proved $A \supset B \in \Gamma$, we get $A \supset B \in \mathcal{C}l^+(\Gamma)$, and this concludes the proof of point (v).

Let $C \in \textsc{Sr}(G)$ such that $\mathcal{K}, \alpha \nVdash C$; we show $C \in \mathcal{C}l^-(\Delta \cup \Lambda)$. The cases $C \in \mathcal{V}$, $C = A \wedge B$ and $C = A \vee B$ can be proved as in the case (C1). Let $C = A \supset B$; we show that $A \supset B \in \Delta \cup \Lambda$. Since $K, \alpha \nVdash A \supset B$, there exists $\gamma \in W$ such that $\alpha \leq \gamma$ and $\mathcal{K}, \gamma \Vdash A$ and $\mathcal{K}, \gamma \nVdash B$. If $\gamma = \alpha$, by (IH2) we get $A \in \mathcal{C}l^+(\Gamma)$ and $B \in \mathcal{C}l^-(\Delta \cup \Lambda)$. By Lemma 3(iii), it follows that $A \supset B \in \Delta$. Let us assume $\alpha < \gamma$. Then, $\beta \leq \gamma$, hence $\mathcal{K}, \beta \nVdash A \supset B$. By (IH1), we get $A \supset B \in \mathcal{C}l^-(\Delta' \cup \Lambda')$, which implies $A \supset B \in \Delta' \cup \Lambda'$. Since $\Delta' \cup \Lambda' \subseteq \Delta$, we get $A \supset B \in \Delta$. Having proved that $A \supset B \in \Delta$, it follows that $A \supset B \in \mathcal{C}l^-(\Delta \cup \Lambda)$, and this concludes the proof of point (vi). □

To conclude the proof of completeness, we need to prove that:

Lemma 6 If $G \notin \mathrm{GD}_k$, then there exists a countermodel $\mathcal{K}$ for G such that $\mathrm{h}(\mathcal{K}) \leq k$ and $\mathcal{K}$ is G-separable.

Proof. We give a sketch of the proof. Let us assume $G \notin \mathrm{GD}_k$. Then, there exists a model $\mathcal{K}_1 = \langle W_1, \leq_1, \rho_1, V_1 \rangle$ such that $\mathcal{K}_1, \rho_1 \nVdash G$ and $\mathrm{h}(\rho_1) \leq k$. We define the countermodel $\mathcal{K}$ in two steps. Firstly, we define the model $\mathcal{K}_2$ obtained from $\mathcal{K}_1$ by adding to each set $V_1(\alpha)$ the propositional variables in $\textsc{Sl}^{\mathrm{At}}(G) \setminus \textsc{Sr}^{\mathrm{At}}(G)$. Secondly, we get $\mathcal{K}$ by filtrating $\mathcal{K}_2$. The model $\mathcal{K}_2 = \langle W_2, \leq_2, \rho_2, V_2 \rangle$ is defined as follows:

$$\begin{array}{l} W_2 \ = \ W_1 \quad \leq_2 = \leq_1 \quad \rho_2 \ = \ \rho_1 \\ \forall \alpha \in W_1,\ V_2(\alpha) \ = \ \Big(V_1(\alpha) \cup (\textsc{Sl}^{\mathrm{At}}(G) \setminus \textsc{Sr}^{\mathrm{At}}(G)) \Big) \setminus (\textsc{Sr}^{\mathrm{At}}(G) \setminus \textsc{Sl}^{\mathrm{At}}(G)) \end{array}$$

By induction on $|C|$, we can prove that:

(1) for every $\alpha \in W_1$ and $C \in \textsc{Sl}(G)$, $\mathcal{K}_1, \alpha \Vdash C$ implies $\mathcal{K}_2, \alpha \Vdash C$;

(2) for every $\alpha \in W_1$ and $C \in \textsc{Sr}(G)$, $\mathcal{K}_1, \alpha \nVdash C$ implies $\mathcal{K}_2, \alpha \nVdash C$.

Let us introduce the following relation between worlds of W_2:

$$\alpha \sim \beta \quad \text{iff} \quad V_2(\alpha) \cap \mathrm{Sf}^{\mathrm{At}}(G) \ = \ V_2(\beta) \cap \mathrm{Sf}^{\mathrm{At}}(G)$$

It is easy to check that:

- $\sim$ is an equivalence relation;
- If $\alpha \leq_2 \beta$ and $\alpha' \sim \alpha$ and $\beta' \sim \beta$ then $\alpha' \sim \beta'$ or $\alpha' <_2 \beta'$.

We turn $\mathcal{K}_2$ into a G-separable model $\mathcal{K}$ by collapsing $\sim$-equivalent worlds. For $\alpha \in W_2$, let $[\alpha]$ denote the equivalence class of α (w.r.t. $\sim$) and let W be the quotient of W_2. By the above properties, the model $\mathcal{K} = \langle W, \leq, \rho, V \rangle$ can be defined as follows:

$$\begin{array}{l} \leq = \{ ([\alpha], [\beta]) \mid \alpha \leq_2 \beta \} \qquad \rho \ = \ [\rho_2] \\ \forall \alpha \in W_2,\ V([\alpha]) \ = \ V_2(\alpha) \cap \mathrm{Sf}^{\mathrm{At}}(G) \end{array}$$

By induction on $|C|$, we can prove that:

(3) For every $\alpha \in W_2$ and $C \in \mathrm{Sf}(G)$, $\mathcal{K}_2, \alpha \Vdash C$ iff $\mathcal{K}, [\alpha] \Vdash C$.

We show that $\mathcal{K}$ is G-separable. Let $[\alpha] < [\beta]$. Then, $\alpha \leq_2 \beta$ and $\alpha \not\sim \beta$. Thus, that there exists $p \in V_2(\beta) \setminus V_2(\alpha)$, and this implies $p \in \textsc{Sl}(G) \cap \textsc{Sr}(G)$. Since $\mathcal{K}_1, \rho_1 \nVdash G$ and $G \in \textsc{Sr}(G)$, by (2) and (3) we get $\mathcal{K}, \rho \nVdash G$, hence $\mathcal{K}$ is a countermodel for G. Finally, we observe that $\mathrm{h}(\mathcal{K}) \leq \mathrm{h}(\mathcal{K}_2) = \mathrm{h}(\mathcal{K}_1) = k$. $\square$

Let us assume $G \notin \mathrm{GD}_k$. By Lemma 6, there exists a model $\mathcal{K} = \langle K, \leq, \rho, V \rangle$ such that $\mathcal{K}, \rho \nVdash G$, $\mathrm{h}(\rho) \leq k$ and $\mathcal{K}$ is G-separable. Let $\mathrm{Sat}_G(\rho) = \Gamma \not\Rightarrow_{k'} \Lambda\,;\,\Delta$. By Lemma 5(i), $k' = \mathrm{h}(\rho) \leq k$ and there exists an $\mathbf{R}_{\textsc{gd}}(G)$-refutation of $\mathrm{Sat}_G(\rho)$. Since $\mathcal{K}, \rho \nVdash G$, by Lemma 5(vi) we get $G \in \mathcal{C}l^{-}(\Delta \cup \Lambda)$. We conclude $\vdash^k_G G$, and this proves the completeness theorem. As a corollary, we get

Theorem 3 $G \notin \mathrm{GD}$ iff there exists an $\mathbf{R}_{\textsc{gd}}(G)$-refutation of G.

6. Conclusions

In this paper we have introduced a forward calculus $\mathbf{R}_{\mathrm{GD}}(G)$ to derive the non-validity of a goal formula G in Gödel-Dummett logics. From an $\mathbf{R}_{\mathrm{GD}}(G)$-refutation of G we can extract a countermodel for G. As for the proof-search strategy, we have presented the naive forward strategy of [15], we leave as future work the investigation of clever strategies (e.g., using subsumption to reduce redundancies as those discussed in [8]) and the implementation of the calculus exploiting the full-fledged Java Framework JTabWb [33]. The refinement of the forward proof-search strategy and the implementation are key step to compare our approach with the ones presented in [34, 9, 10]. We also aim to extend our approach to other intermediate logics.

References

[1] S. Feferman, D. J.W., S. Kleene, G. Moore, R. Solovay, J. van Heijenoort (Eds.), Kurt Gödel: Collected Works. Vol. 1: Publications 1929-1936, Oxford University Press, Inc., 1986.

[2] M. Dummett, A propositional calculus with a denumerable matrix, Journal of Symbolic Logic 24 (1959) 96–107.

[3] P. Hájek, Metamathematics of Fuzzy Logic, volume 4 of *Trends in Logic*, Kluwer, 1998.

[4] F. Aschieri, A. Ciabattoni, F. A. Genco, Gödel logic: From natural deduction to parallel computation, in: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017, IEEE Computer Society, 2017, pp. 1–12. URL: https://doi.org/10.1109/LICS.2017.8005076. doi:10.1109/LICS.2017.8005076.

[5] A. Avron, Hypersequents, logical consequence and intermediate logics for concurrency, Annals for Mathematics and Artificial Intelligence 4 (1991) 225–248.

[6] F. Aschieri, On Natural Deduction for Herbrand Constructive Logics I: Curry-Howard Correspondence for Dummett's Logic LC, Logical Methods in Computer Science 12 (2016) 1–31.

[7] A. Beckmann, N. Preining, Hyper Natural Deduction for Gödel Logic—A natural deduction system for parallel reasoning, Journal of Logic and Computation 28 (2018) 1125–1187.

[8] C. Fiorentini, M. Ferrari, Duality between unprovability and provability in forward refutation-search for intuitionistic propositional logic, ACM Transactions Computational Logic (TOLC) 21 (2020) 22:1–22:47.

[9] G. Fiorino, Terminating calculi for propositional Dummett logic with subformula property, Journal of Automated Reasoning 52 (2014) 67–97.

[10] D. Larchey-Wendling, Gödel-dummett counter-models through matrix computation, Electronic Notes Theoretical Compututer Science 125 (2005) 137–148.

[11] M. Ferrari, C. Fiorentini, G. Fiorino, Forward countermodel construction in modal logic K, in: P. Felli, M. Montali (Eds.), CILC 2018, volume 2214 of *CEUR*, CEUR-WS.org, 2018, pp. 75–81.

[12] C. Fiorentini, M. Ferrari, A forward unprovability calculus for intuitionistic propositional logic, in: R. A. Schmidt, C. Nalon (Eds.), TABLEAUX 2017, volume 10501 of *LNCS*, Springer, 2017, pp. 114–130.

[13] C. Fiorentini, M. Ferrari, A forward internal calculus for model generation in S4, Journal of Logic and Computation 31 (2021) 771–796.

[14] S. J. Maslov, An invertible sequential version of the constructive predicate calculus, Zap. Naučn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI) 4 (1967) 96–111.

[15] A. Degtyarev, A. Voronkov, The inverse method, in: J. R. et al. (Ed.), Handbook of Automated Reasoning, Elsevier and MIT Press, 2001, pp. 179–272.

[16] K. Chaudhuri, F. Pfenning, G. Price, A logical characterization of forward and backward chaining in the inverse method, in: U. F. et al. (Ed.), IJCAR 2006, volume 4130 of *LNCS*, Springer, 2006, pp. 97–111. URL: http://dx.doi.org/10.1007/11814771_9. doi:10.1007/11814771_9.

[17] K. Donnelly, T. Gibson, N. Krishnaswami, S. Magill, S. Park, The inverse method for the logic of bunched implications, in: F. B. et al. (Ed.), LPAR 2004, volume 3452 of *LNCS*, Springer, 2004, pp. 466–480. URL: http://dx.doi.org/10.1007/978-3-540-32275-7_31. doi:10.1007/978-3-540-32275-7_31.

[18] L. Kovács, A. Mantsivoda, A. Voronkov, The inverse method for many-valued logics, in: F. C. et al. (Ed.), MICAI 2013, volume 8265 of *LNCS*, Springer, 2013, pp. 12–23. URL: http://dx.doi.org/10.1007/978-3-642-45114-0_2. doi:10.1007/978-3-642-45114-0_2.

[19] T. Dalmonte, B. Lellmann, N. Olivetti, E. Pimentel, Countermodel construction via optimal hypersequent calculi for non-normal modal logics, in: S. N. Artëmov, A. Nerode (Eds.), LFCS 2020, volume 11972 of *LNCS*, Springer, 2020, pp. 27–46.

[20] T. Dalmonte, N. Olivetti, G. L. Pozzato, HYPNO: theorem proving with hypersequent calculi for non-normal modal logics (system description), in: N. Peltier, V. Sofronie-Stokkermans (Eds.), IJCAR 2020, volume 12167 of *LNCS*, Springer, 2020, pp. 378–387.

[21] M. Ferrari, C. Fiorentini, G. Fiorino, Contraction-free linear depth sequent calculi for intuitionistic propositional logic with the subformula property and minimal depth counter-models, Journal of Automated Reasoning 51 (2013) 129–149. doi:10.1007/s10817-012-9252-7.

[22] C. Fiorentini, Terminating sequent calculi for proving and refuting formulas in S4, Journal of Logic and Computation (2012).

[23] C. Fiorentini, An ASP approach to generate minimal countermodels in intuitionistic propositional logic, in: S. Kraus (Ed.), IJCAI, ijcai.org, 2019, pp. 1675–1681.

[24] D. Galmiche, D. Méry, Proof-search and countermodel generation in propositional BI logic, in: N. Kobayashi, B. C. Pierce (Eds.), TACS 2001, volume 2215 of *LNCS*, Springer, 2001, pp. 263–282.

[25] D. Galmiche, D. Méry, Resource graphs and countermodels in resource logics, Electronic Notes in Theoretical Computer Science 125 (2005) 117–135.

[26] L. A. Nguyen, Constructing finite least Kripke models for positive logic programs in serial regular grammar logics, Logic Journal of the IGPL 16 (2008) 175–193.

[27] L. Pinto, R. Dyckhoff, Loop-free construction of counter-models for intuitionistic propositional logic, in: B. et al. (Ed.), Symposia Gaussiana, Conference A, Walter de Gruyter, Berlin, 1995, pp. 225–232.

[28] N. Sara, Proofs and countermodels in non-classical logics, Logica Universalis (2014) 1–36.

[29] T. Skura, Refutation Methods in Modal Propositional Logic, Semper Warsaw, 2013.

[30] A. Troelstra, H. Schwichtenberg, Basic Proof Theory, volume 43 of *Cambridge Tracts in*

Theoretical Computer Science, 2ed ed., Camb. Univ. Press, 2000.

[31] A. Chagrov, M. Zakharyaschev, Modal Logic, Oxford University Press, 1997.

[32] F. Baader, T. Nipkow, Term rewriting and all that., Cambridge University Press, 1998.

[33] M. Ferrari, C. Fiorentini, G. Fiorino, JTabWb: a Java framework for implementing terminating sequent and tableau calculi, Fundamenta Informaticae 150 (2017) 119–142.

[34] C. Fiorentini, M. Ferrari, SAT-based proof search in intermediate propositional logics, in: J. B. et al. (Ed.), IJCAR 2022, volume 13385 of *LNAI*, 2022, pp. 57–74.

Is This My Place? A Twitter Dataset To Recognize The Sense Of Place

Giovanni Siragusa[1], Livio Robaldo[2], and Luigi Di Caro[1]

[1] Computer Science Department, Universitá degli Studi di Torino, Italy
{siragusa, dicaro}@di.unito.it
[2] Legal Innovation Lab Wales, Swansea University
livio.robaldo@swansea.ac.uk

Abstract. The concept of *Place* in Geography and related fields has a complex definition involving different facets, ranging from its aspect and setting (e.g., buildings, shops, and the surrounding environment), to its functional semantics and to the sentiments it may evoke. Over the last years, researchers focused on identifying explicit and hidden semantics behind places and their reciprocal similarities. In this article, we used the Cresswell's definition of *place*, which takes into account the sentiments people feel about it, i.e., the so-called *Sense Of Place* (SOP). As our contribution, we first constructed a novel Twitter-based dataset by harvesting tweets referring to 4 different cities having English as their mother-tongue language (New York, San Francisco, Wellington and Sydney). Then, the collected tweets have been labelled by three annotators in terms of expressed SOP. Finally, we used the constructed dataset to train classical Machine Learning (ML) models, evaluating them on a novel (test) set of annotated tweets regarding the city of London. Results demonstrate the validity of the SOP-based data construction in terms of both human agreements and ML accuracy levels.

Keywords: Place · Twitter · Sentiment Analysis

1 Introduction

Artificial Intelligence (AI) is experiencing growth in activity, both in academia and in industry. Current AI-based technologies mostly use Machine Learning (ML) to process documents and replicate legal decision-making.

Although ML provides valid solutions, its overall usefulness is limited in that it tends to disregard specific semantic aspects and legal reasoning.

ML is based on *statistical reasoning*: new cases are classified by similarity with the cases included in the training set. As a result, performance are intrinsically limited. Furthermore, and most important of all, as it is well-known ML tends to behave like a "black box" unable to explain its decisions and it can therefore lead to biases and other discriminatory outcomes: ML trained on biased datasets tend to replicate the same biases on new inputs.

In order to overcome the limits of ML, lot of recent research has been devoted to investigate approaches in symbolic AI. The idea is to plug into the

ML-based system human-understandable symbols, i.e., concepts and other logical constructs, that enable forms of *logical reasoning*. Besides leading to an improvement in the performance, symbolic AI is the path to explainability, needed to contrast biases in decision-making [22].

Nevertheless, as it is well-known symbolic AI is requires lot of efforts, as (human) domain experts must be involved in the creation of symbolic representations such as ontologies or logical rules. Concept are highly *context-sensitive*: their meaning can vary depending on the context in which they are used so that even domain experts can struggle to find an agreement among their definitions. This is particularly true for "general-purpose" abstract concepts such as the concept of "Place", the one on which this paper focuses on (see 1.1 below).

Therefore, the transition from ML-based systems to sustainable approaches in AI fully based on computational logic is still a very long journey. To underpin the transition, lot of contemporary approaches propose to use concepts *in combination with* the statistical inferences of ML [5]. The research presented in this paper is one of these approaches: it presents a ML-based analysis of Twitter comments centered on the concept of "Place".

1.1 The concept of "Place"

Place is a concept frequently mentioned in every-day life. We use it in sentences as *"This is my favourite place"*, *"I finally found my place in the world"*, or *"Lay a place at the table for Mr. T"*. In the commonsense language, the term *place* is used to refer to a city (e.g., New York), a public space (e.g., Central Park), a shop or even to the seat we usually take at the table. These examples suggest that the concept of *place* is broad and ranges from a punctual space to a wide area.

In the Geography domain, the concept of *place* assumes a more structured representation. In particular, according to Cresswell [12,11] this concept involves three different aspects:

- **Location:** the physical absolute point in the space, identified by a set of coordinates;
- **Locale:** the visible features and settings of a place, such as streets, shops, parks and so on;
- **Sense Of Place:** the set of emotions and feelings that a place inspires in people. These sentiments can be subjective when they are based on someone's personal biography, or they can be shared when a group of people feels the same sentiment towards a place.

According to the intuition of Cresswell, a complete definition of *place* asks for a systematic analysis of all the three aspects listed above. This analysis starts with the identification of a place to focus on, and, secondly, the collection of the features and settings that constitute the place of interest and that could have an impact on how people feel it, such as shops, house styles and streets. Such analysis could be conducted both at micro-level, analyzing a single street or house, or at macro-level, analyzing an entire district or city. Finally, the *Sense Of Place* (SOP) can be "extracted" by analyzing a large set of observations by the people living and experiencing the chosen place.

Nowadays, social networks are part of our everyday life, and people are used to disclose their opinions, activities and emotions through them. Therefore, scientific community started to consider people in social networks as social sensors for different fields such as politics, economics and sociology [18]. Thus, it is possible to harvest and analyze their posts to discover high-level aspects. Moreover, the possibility to associate a geographical reference to a post (the so called geotags) or to infer the location starting from significant hashtags allowed scientists to develop map-based data analysis which can also be used to identify disaster-affected areas or regions with high crime rates, as respectively in the works of Cerutti et al. [9] and Ristea et al. [17].

Since users' posts contain both textual and location (the geotag) information, they suit well to define the SOP of a chosen place. Thus, following the idea of Siragusa and Leone [19], in this article we present a dataset of user posts, adopting Twitter as our source of information. Such dataset could be used to train a classifier in order to discern tweets that express a general sentiment from those expressing a sentiment towards a place (i.e., the SOP). Our idea is that an automatic classifier of SOP messages will allow to swiftly obtain large quantity of data, enabling geographers and data-analysts to understand the relationship between people and places and exploring the process that shapes those relationships. For constructing the dataset, we chose four different cities (San Francisco, Wellington, Sydney, and New York) that have English as their mother-tongue language and we collected tweets referring to them. We then asked to three annotators to judge whether the tweets were expressing some SOP with respect to the selected cities. Finally, we trained 4 different state-of-the-art Machine Learning classifiers: Support Vector Machine (SVM), Naive Bayes (NB), Decision Trees (DT) and AdaBoost (AB) to recognize the SOP-tweets, showing good performance especially with SVM.

The paper is structured as follows: Section 2 describes how the tweets were collected and labelled to create the dataset; Section 3 describes how we used the dataset to train the classifiers. Section 4 contains related works concerning the extraction, labelling and analysis of *place*-based data. Section 5 concludes the article with some final remarks.

2 Tweets Extraction

For creating our dataset, we collected from Twitter[3] those tweets referring to the following 4 cities: New York, San Francisco, Wellington and Sydney. We used the Tweepy[4] library for the extraction, and we collected only the tweets containing an hashtag that represented one of the cities of interest. For each city, the hashtags we looked for were:

[3] https://twitter.com
[4] https://www.tweepy.org

New York: #NewYork, #NY, #newyork;
San Francisco: #SanFrancisco, #SF, #sanfrancisco;
Wellington: #Wellington, #wellington;
Sydney: #Sydney, #NSW, #sydney.

We collected a total of more than 2 millions of tweets, even if with a varied distribution over the 4 cities. Then, we applied to each tweet the sentiment classifier of TextBlob[5] (because the SOP is strictly entwined with the sentiment), obtaining its sentiment expressed in the range [-1.0, 1.0], with 0 indicating a neutral tweet. Using the sentiment scale, we filtered out all the tweets that did not express a (positive or negative) sentiment, i.e only those with a zero score. We decided to keep the tweets that are close to zero (e.g., 0.2) since they may express a SOP. Table 1 reports the number of remaining tweets for each city.

City	# Total Tweets	# Tweets with sentiment
San Francisco	10,631	1043
New York	1,274,647	422
Sydney	24,761	1223
Wellington	1,003,007	518
Total	2,313,046	3206

Table 1. The table reports the number of collected tweets of each city and the number of tweets expressing some sentiment (positive or negative).

We then asked to three annotators to make a judgement, under the form of Y (corresponding to *Yes*) and N (corresponding to *No*), to find the tweets expressing a SOP. In particular, the annotators had to follow the following guidelines:

- a tweet is tagged with the label Y if it expresses a feeling, an emotion or a sentiment (either positive or negative) towards a city or an area inside it (e.g., a neighborhood or a park);
- a tweet is tagged with the label N if it does not express any feeling, emotion or sentiment (either positive or negative) towards a city or an area inside it;
- a tweet that seems to have been written to promote trips to a city or to promote tourist activities should not be tagged with Y since it does not correspond to a sincere emotion, having instead only advertising purposes;
- a tweet expressing a political exposure towards the people who govern the city does not express a feeling towards the city itself, so it should be labelled with N.

We computed the inter-annotators agreement using Fleiss' kappa coefficient [13], and the agreement of each annotators pair using Cohen's kappa coefficient [10]. Figure 1 reports Fleiss's kappa on each city, where San Francisco resulted to be the city with the highest agreement (close to 0.7) between the three annotators

[5] https://textblob.readthedocs.io/en/dev/

according to the SOP expressed by the corresponding tweets. Figures 2, 3 and 4 report Cohen's kappa coefficient for each couple of annotators. From these, we can notice that Annotator 1 and Annotator 3 had high agreement, in contrast with Annotator 2.

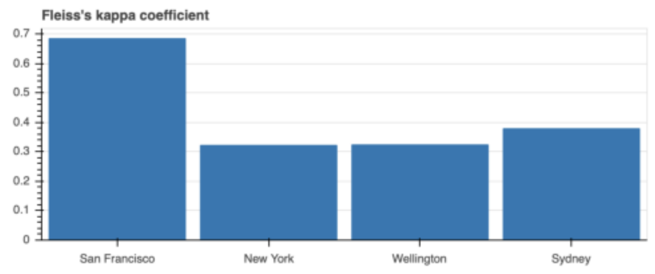


Fig. 1. The images shows the Fleiss' kappa coefficient for all the four cities.

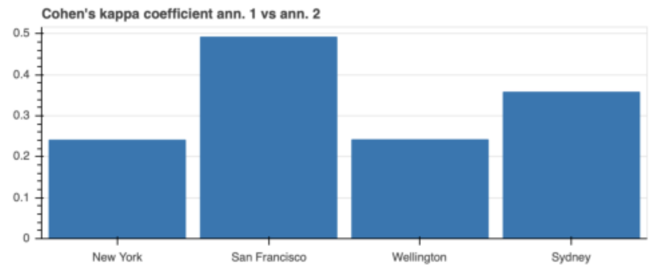


Fig. 2. The image shows the Cohen' Kappa for annotator 1 and annotator 2.

We assigned the label Y or N with each tweet using a majority vote scheme (i.e., if two annotators expressed Y on the same tweet, we labelled it with Y to indicate a SOP, and viceversa). Table 2 shows some tweets with their annotation. The dataset can be found at the following link: https://bit.ly/3s5KF58.

Table 3 reports the number of tweets of each city that have a sentiment, and how many of them actually express a SOP after the application of the majority vote scheme.

Finally, we analyzed the most frequent words in both SOP and non-SOP tweets. More in detail, we applied the following pre-processing pipeline: first, we removed URLs, names (words starting with @) and ReTweets (RT) tags, keeping only those hashtags linking to salient information; then, we lowercased all words and stemmed them, using the NLTK library[6]. We then generated a wordcloud

[6] https://www.nltk.org

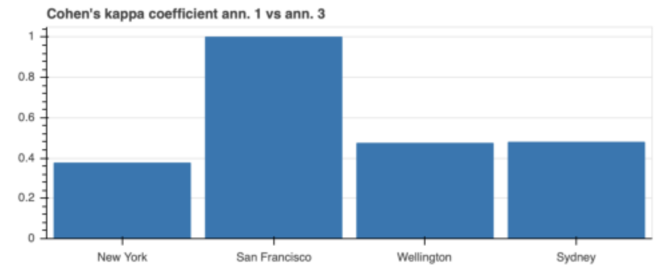


Fig. 3. The image shows the Cohen' Kappa for annotator 1 and annotator 3.

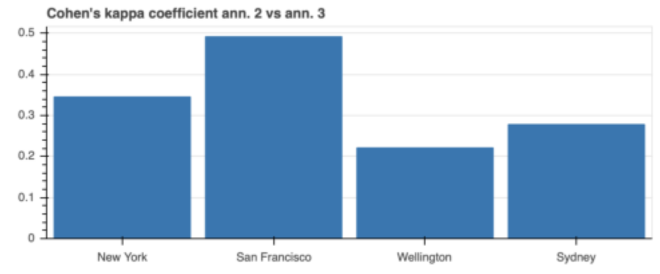


Fig. 4. The image shows the Cohen' Kappa for annotator 2 and annotator 3.

using the library WordCloud[7]. Figures 5 and 6 show the most frequent words for both kinds of tweets. Comparing the two figures, we can see that both contain words conveying sentiments as *love*, *amazing*, *happy*, *beautiful*, and so forth. However, SOP-tweets also contain words such as *love people*, *people living*, *place*, *spot* and *live*, highlighting how those tweets are more focused on the city, its districts and citizens; non-SOP tweets, instead, show more general sentiments.

3 Automatic Detection of SOP

In this article, one contribution regards the idea of automatically distinguishing tweets expressing a general sentiment from those having a Sense of Place (SOP).

We used the dataset described in Section 2 to perform two types of evaluation. To check if it is possible to train a classifier to recognize other tweets that express a SOP, we performed a 5-fold cross evaluation training the classifier on 4 folds and testing on the remaining one. We processed each tweet in the following way: first, we removed names (words having @ in the tweets), URLs, and RT (ReTweetted) tags, leaving the hashtags since they sometimes express a sentiment towards a place; finally, we lowercased all words. The resulting tweets are then passed to a pre-trained Sentence-Bert [16] model in order to obtain their vector representation, which is used to train these classifiers:

[7] https://amueller.github.io/word_cloud/

Tweet	Label
Great weekend with Hannah Nguyen-Chang in #SF. Think we overdone it this Saturday as she's experiencing lots of pa...	SOP
We are on sale in #SanFrancisco Access Hollywood Live (NBC) says "You HAVE to check them out! That was tremendous!"	non-SOP
From https://t.co/MDjtLDWTeX - One of the greatest things about #Wellington is that it's full of art. As Jeff Tweedy of @people said, "I think art is a consolation regardless of its content. It has the power to move and make you feel like you're not."	SOP
Hype is getting hyped this weekend! We are so grateful to have fantastic customers in #Newyork, #Sanfrancisco and...	non-SOP

Table 2. The table reports some tweets with their label.

City	#Tweets	#SOP-Tweets
San Francisco	1043	171
New York	422	7
Sydney	1223	73
Wellington	518	42
Total	3206	293

Table 3. The table reports, for each city, the number of tweets expressing a sentiment and the number of tweets that contain a Sense Of Place.

- a Support Vector Machine (SVM) with radial basis function kernel;
- a Gaussian Naive Bayes (NB);
- a Random Forest (RF) classifier with a minimum of five examples for each leaf. It has also the advantage of explaining why a tweet is classified as SOP or non-SOP since each node of the tree is composed of simple human-readable rules;
- and an AdaBoost (AB) classifier.

For each classifier, we performed a grid search to find the optimal parameters, using the Scikit-learn[8] implementation of the classifiers.

We compared them with two baseline: i) a random classifier which randomly assigns the label SOP and non-SOP to tweets; and ii) a majority class classifier which assigns the majority class (non-SOP label) to all tweets. Table 4 reports the Accuracy, Precision, Recall and F-measure of each classifier. We also considered the dataset unbalance in computing the metrics. From the table, we can notice that both SVM and RF are mostly able to recognize SOP-tweets.

We also tested the classic Term Frequency - Inverse Document Frequency (TF-IDF) vectorization to train the classifiers, adding two further steps to the cleaning pipeline: stopwords removal and word stemming. The classifiers trained with this latter vectorization performed worst on all metrics, loosing about 2

[8] https://scikit-learn.org/stable/

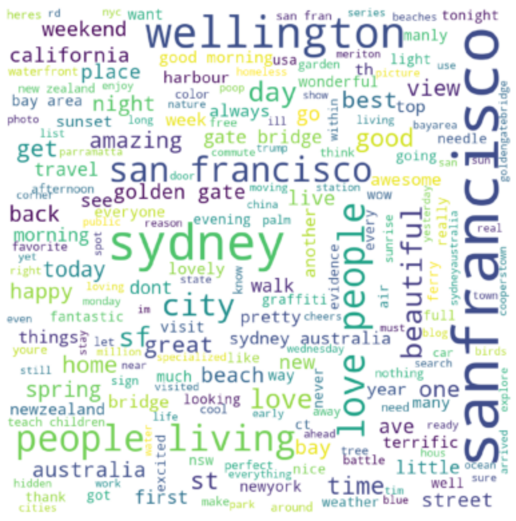


Fig. 5. The image shows a wordcloud of the tweets that contain a Sense Of Place.

Classifier	Accuracy	Precision	Recall	F-measure
Random	50.0	50.0	50.0	50.0
Majority Class	0.17	0.001	0.17	0.003
SVM	**83.45**	**83.80**	**81.39**	**83.45**
Gaussian NB	79.35	79.75	79.36	79.28
AB	75.08	75.17	75.09	75.06
RF	80.71	80.97	80.72	80.67

Table 4. The table reports the cross-fold evaluation of the classifiers.

percentage points for SVM and RF, and about 4 percentage points for Gaussian NB and AB.

The problem of the previous evaluation is that we do not know how well a classifier trained on the dataset will perform on new tweets. For this reason, we collected an additional set of tweets about the city of London using the following hashtags: *#London*, *#LND* and *#london*. Following the same steps explained in Section 2, we extracted only those having a sentiment, obtaining a dataset composed of 1388 tweets. We then asked to the three annotators to judge them following the guidelines provided for the previous labelling task. After applying the majority voting scheme, we obtained that only 73 tweets expressed a SOP. The Fleiss' kappa coefficient [13] for the three annotators was a bit lower, i.e. 0.34, with a Cohen's kappa of 0.4 for Annotators 1 and 3, 0.15 for Annotators 1 and 2, and 0.23 for Annotators 2 and 3.

Finally, we trained the classifiers on the 4-cities dataset and then tested them on the new London tweets. Table 5 reports the results, where the Majority Class

Fig. 6. The image shows a wordcloud of non-SOP tweets.

classifier shows the highest accuracy due to the high unbalance of the dataset towards the non-SOP label.

We can notice that the Gaussian NB classifier, despite ranked third in the previous evaluation, performs well in the London dataset, meaning that it is able to transfer its knowledge on SOP tweets on other datasets; AB classifier, instead, has poor performance on both datasets, demonstrating to be not suitable for this task. Finally, the SVM classifier performs well against the other ones for both datasets; however, it has a drop in performance for SOP-tweets only, obtaining 21.78 for the Precision, 84.72 for the Recall and 34.66 for the F-measure. Using the TF-IDF vectorization, we noticed a drop in the F-measure of 2 percentage points for all classifiers compared to the ones of Table 5. These results demonstrate the complexity of the SOP classification task, being more fine-grained with respect to the standard sentiment detection.

Classifier	Accuracy	Precision	Recall	F-measure
Random	50.0.	50.0.	50.0	50.0
Majority Class	**94.8**	47.4	50.0	48.7
SVM	83.41	60.39	**84.03**	62.58
Gaussian NB	86.17	**61.22**	81.54	**64.33**
AB	75.19	56.92	78.38	55.33
RF	84.57	60.08	80.04	62.41

Table 5. The table reports the results on London dataset.

We then analyzed the errors made by the SVM classifier (false positives and false negatives). We found that the false positives are misclassified due to the cleaning phase. In detail, since we clean the tweet removing names and RT tags, the resulting text could resemble the one of a SOP-tweet, especially in the case of TF-IDF vectors where the stopwords are removed. Table 6 reports some examples.

Tweet	Clean Text	TF-IDF's Bag-Of-Words
Loved every minute of being on the #panel - what an amazing bunch!! @UELAlumni always delivers the best!! #student #alumni #university #London	`loved every minute of being on the #panel - what an amazing bunch ! ! always delivers the best ! ! #student #alumni #university #london`	`love everi minut #panel amaz bunch alway deliv best #student #alumni #univers #london`
#LONDON: Thanks London its been emotional! See you all again next year #WeRunThisCity	`#london : thanks london its been emotional ! see you all again next year #werunthiscity`	`#london thank london emot see next year #werunthisc`
#london #londonpride highly appreciated the hospitality! See you soon #england	`#london #londonpride highly appreciated the hospitality ! see you soon #england`	`#london #londonprid highli appreci hospit see soon #england`

Table 6. The table reports some tweets of the London dataset classified as false positive, and their pre-processed text passed to Bert.

Concerning the false negatives, this is the most interesting case. We found two types of error (reported in Table 7):

Annotation error: there are tweets that have been classified as SOP in the dataset, but they do not contain a sentiment towards a place; thus the classifier correctly recognized them. These are borderline cases, where the sentiment expressed by the user regards some activity they performed (or attended) at the city; for instance, the tweet "*It's going to be a great weekend @LondonTattooCon look forward to seeing the best #tattooartist*" has been labelled as SOP by the annotators, but it expresses a positive sentiment regarding the intention to attend the tattoo convention. Since these tweets are complex cases, our objective as future work is to discuss with annotators and linguistic experts in order to improve the guidelines and the quality of the dataset;

Classification error: similarly to the false positive cases, some tweets resemble the negatives ones because they do not have any word expressing a clear sentiment towards a place.

Tweet	Error type
It could be a magical place to go if you fancy a cheeky Butterbeer or some Pumpkin Juice. #HarryPotter	classification
Lovely morning to start the 2nd #DevOpsDays in #London as we talk all things #DevOps again.	annotation
Spent a great #tabletopRPG evening NMDLondon tonight Too many AWESOME GMs to fit in one video #NoMoreDamsels #Dung...	annotation

Table 7. The table reports some tweets of the London dataset classified as false negative.

4 Related Works

The specification of a geographical reference in a shared post is nowadays an habit for the users of the most famous social networks as *Twitter*[9], *Facebook*[10] and *Instagram*[11]. The geographical references can appear in a post either in the form of a geotag or an hashtag that expresses the name of the location (a city, a restaurant, a public park, etc.). Such information have been used by scientists in order to develop geographical and temporal based studies in the field of Geography and Data Science. In addition to these well-known platforms, other new location-based services emerged in the last years. Among them, we mention *Trendsmap*[12] which shows on a map the latest trends emerging from Twitter, *Ushahidi*[13] which collects and visualises information about crisis witnesses providing the users the possibility to respond and *FixMyStreet*[14] which allows the UK citizens to signal streets problems (pot holes, unsafe walls, not working lampposts) to the local authorities. *FirstLife*[15] [3] is a more interactivity-oriented service which focuses its attention on the user intended as citizen, giving them the possibility to interact with a map on which they can share events, news and even aggregate people. Moreover, the data are associated with a temporal dimension which allows users to filter and order the information according to time.

In the context of geo-oriented social networks, it is easy to define the concept of place expressed by Cresswell [11,12]. Both *location* and *locale* are coded in the geographical map showed to the user, while the *sense of place* (SOP) is expressed by the user via posts. Furthermore, the concept of place has evolved together with technology, becoming the conceptual fusion of space and experience [15]. The advent of Covid-19 has fasted the projection of the concept of place from the physical world to the virtual one, since our freedoms of movement were taken away [6]; For instance, virtual meeting allowed to gaze the interior of our boss' (or employee) house, disclosing aspects of one another's home that reveal

[9] www.twitter.com
[10] www.facebook.com
[11] www.instagram.com
[12] www.trendsmap.it
[13] www.ushahidi.com
[14] www.fixmystreet.com
[15] www.firstlife.org

the sense of place [14]. For these reasons, we decided to use Twitter as source to extract SOP messages, where the hashtag of a city (e.g., #tokyo) defines the space boundaries of the place, and the post content its sense. Our work is similar to Siragusa and Leone [20], where the authors used Latent Dirichlet Allocation [7] (LDA) to extract those tweets that contain the sense of place. Instead of applying the LDA model, we decided to create a SOP dataset by hiring annotators; such dataset could be used to train a classifier to recognize SOP tweets or to conduct data-analysis.

Other works explored both the use of Twitter and the concept of place. Besbris et al. [4] studied the spatial stigma, i.e. how the neighbours of a place perceive it in a negative way due to crime, disorders, poverty and even racial isolation; and the residents of such place may embody the negative characteristics. Zhu et al. [23] tried to quantify the semantic of places according to their interaction with streets. They found that the interaction is beneficial to identify the semantic. Sakaki et al. [18] used the intuition of considering the users as social sensors in order to implement event detection. Cataldi et al. [8] extracted in real time the most emerging topics expressed by the community based on the interests of a specific user in a particular temporal frame, while Allisio et al. [2] exploited the temporal and spacial information associated with the tweets in order to produce a daily estimation of the degree of happiness of the main Italian cities. Adams et al. [1] developed a system based on Latent Dirichlet Allocation [7] (LDA) to extract place characteristics from a travel blog. They also correlated those characteristics to find similar places. Steiger et al. [21] applied LDA on collected tweets to extract real-world characteristics. They also tracked the topics frequency along a week, finding that people tend to talk about *home* and *work*.

5 Conclusion

In this article, we presented a dataset of tweets that express a Sense Of Place (SOP). We also showed that is possible to train a classifier on such dataset to label new tweets.

We started from the idea presented in Siragusa and Leone [19], where the authors used Twitter as a resource to collect users' posts and then find those ones that express a SOP. Differently from them, we employed three annotators to label all the tweets. Then, we created a dataset taking those tweets labelled as expressing or not a SOP related to a city of reference. Finally, we trained four Machine Learning classifiers, showing both it is possible to automatically recognize the SOP and that it is a difficult task. However, as explained in the introduction, most of the tested state-of-art Machine Learning classifiers behave as "black boxes", being unable to explain why a tweet has been classified with a SOP (or non-SOP) label. The only exception is the Random Forest, which produces simple rules (e.g., length of the sentence greater than a threshold) connected via conjunction or disjunction logical operators. Although these rules can be read by human experts, they tend to be too specific or situational, making difficult to generalize them.

As future work, we would like to extend the size of the dataset in order to train Neural Networks models. We think that these models could suit well for this kind of task since they are capable to deeply analyzing the meaning of each word of the sentence and its interaction with other ones. A preliminary result of the positive impact of these models has been showed by the Bert vectorization of the tweets. Another research path is the adoption of symbolic AI in conjunction with Machine Learning models to improve both their accuracy and explainability. In this way, the rules defined by the models could be used by data-analysts and geographers to explain why a tweet has been labelled with a certain tag. Furthermore, such rules will allow to understand the errors performed by the classifier, either improving it or the annotation process by refining the guidelines.

References

1. Adams, B., McKenzie, G.: Inferring thematic places from spatially referenced natural language descriptions. In: Crowdsourcing geographic knowledge, pp. 201–221. Springer (2013)
2. Allisio, L., Mussa, V., Bosco, C., Patti, V., Ruffo, G.F.: Felicittà: Visualizing and estimating happiness in italian cities from geotagged tweets. In: CEUR WORKSHOP PROCEEDINGS. vol. 1096, pp. 95–106. CEUR Workshop Porceedings (2013)
3. Antonini, A., Boella, G., Buccoliero, S., Calafiore, A., Di Caro, L., Giorgino, V., Ruggeri, A., Salaroglio, C., Sanasi, L., Schifanella, C.: First life, from the global village to local communities. In: 1st IASC Thematic Conference on Urban Commons (2015)
4. Besbris, M., Faber, J.W., Rich, P., Sharkey, P.: The geography of stigma: Experimental methods to identify the penalty of place. In: Audit studies: Behind the scenes with theory, method, and nuance, pp. 159–177. Springer (2018)
5. Bibal, A., Lognoul, M., de Streel, A., Frénay, B.: Legal requirements on explainability in machine learning. Artificial Intelligence and Law **29**(2) (2021)
6. Bissell, D.: A changing sense of place: Geography and covid-19. Geographical Research **59**(2), 150–159 (2021)
7. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. Journal of machine Learning research **3**(Jan), 993–1022 (2003)
8. Cataldi, M., Caro, L.D., Schifanella, C.: Personalized emerging topic detection based on a term aging model. ACM Transactions on Intelligent Systems and Technology (TIST) **5**(1), 7 (2013)
9. Cerutti, V., Fuchs, G., Andrienko, G., Andrienko, N., Ostermann, F.: Identification of disaster-affected areas using exploratory visual analysis of georeferenced tweets: application to a flood event. Association of Geographic Information Laboratories in Europe: Helsinki, Finland p. 5 (2016)
10. Cohen, J.: A coefficient of agreement for nominal scales. Educational and psychological measurement **20**(1), 37–46 (1960)
11. Cresswell, T.: Place. In: International Encyclopedia of Human Geography. vol. 8, pp. 169–177. Elsevier (2009)
12. Cresswell, T.: Place–part i. The Wiley-Blackwell companion to human geography pp. 235–244 (2011)

13. Fleiss, J.L.: Measuring nominal scale agreement among many raters. Psychological bulletin **76**(5), 378 (1971)
14. Hay, I.: Zoom and place: Video conferencing and virtual geography. South Australian Geographical Journal **116**(1), 7–11 (2020)
15. Merschdorf, H., Blaschke, T.: Revisiting the role of place in geographic information science. ISPRS International Journal of Geo-Information **7**(9), 364 (2018)
16. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics (11 2019), `http://arxiv.org/abs/1908.10084`
17. Ristea, A., Kurland, J., Resch, B., Leitner, M., Langford, C.: Estimating the spatial distribution of crime events around a football stadium from georeferenced tweets. ISPRS International Journal of Geo-Information **7**(2), 43 (2018)
18. Sakaki, T., Okazaki, M., Matsuo, Y.: Earthquake shakes twitter users: real-time event detection by social sensors. In: Proceedings of the 19th international conference on World wide web. pp. 851–860. ACM (2010)
19. Siragusa, G., Leone, V.: Such a wonderful place: extracting sense of place from twitter. In: European Semantic Web Conference. pp. 397–405. Springer (2018)
20. Siragusa, G., Leone, V.: Such a wonderful place: extracting sense of place from twitter. In: European Semantic Web Conference. pp. 397–405. Springer (2018)
21. Steiger, E., Westerholt, R., Resch, B., Zipf, A.: Twitter as an indicator for whereabouts of people? correlating twitter with uk census data. Computers, Environment and Urban Systems **54**, 255–265 (2015)
22. Waltl, B., Vogl, R.: Explainable artificial intelligence - the new frontier in legal informatics. Jusletter IT. **22** (2018)
23. Zhu, R., McKenzie, G., Janowicz, K.: Are streets indicative of place types? (2019)

Epistemic Logic Programs: an Approach to Semantic Comparison[⋆]

Stefania Costantini[1,3], Andrea Formisano[2,3,*]

[1]DISIM - Università dell'Aquila, via Vetoio–loc. Coppito, 67100 L'Aquila, Italy
[2]DMIF - Università di Udine, via delle Scienze 206, 33100 Udine, Italy
[3]GNCS-INdAM, piazzale Aldo Moro 5, 00185 Roma, Italy

Abstract

Epistemic Logic Programs (ELPs) extend Answer Set Programming (ASP) with epistemic operators. The semantics of such programs is provided in terms of *world views*, which are sets of belief sets. Several semantic approaches have been proposed over time to characterize world views. Recent work has introduced semantic properties that should be met by any semantics for ELPs. We propose a new method, easy but, we believe, effective, to compare the different semantic approaches.

Keywords

Answer Set Programming, Epistemic Logic Programs, ELP semantics

1. Introduction

Epistemic Logic Programs (ELPs, in the following just 'programs' if not explicitly stated differently), were first introduced in [1, 2], and extend Answer Set Programs (ASP programs), defined under the Answer Set Semantics of [3], with *epistemic operators* that are able to introspectively "look inside" a program's own semantics, which is defined in terms of its "answer sets". In fact, $\mathbf{K}A$ means that the (ground) atom A is true in every answer set of the very program Π where $\mathbf{K}A$ occurs, whereas $\mathbf{M}A$ means that A is true in some of the answer sets of Π. The *epistemic negation operator* **not** A expresses that A *is not provably true*, meaning that A is false in at least one answer set of Π. It is easy to see that the operators are interchangeable, as $\mathbf{M}A$ can be defined as $not\,\mathbf{K}not\,A$, and **not** A as $not\,\mathbf{K}A$, not being standard ASP default negation.

Semantics of ELPs is provided in terms of *world views*: instead of a unique set of answer sets like in Answer Set Programming (ASP), there is now a set of such sets. Each world view consistently satisfies (according to a given semantics) the epistemic expressions that appear in a given program. Many semantic approaches for ELPs have been introduced beyond the seminal one of [1], among which we mention [4, 5, 6, 7, 8, 9, 10].

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
[⋆]Research partially supported by Action COST CA17124 "DigForASP" and by projects INDAM GNCS-2020 *NoRMA* and INDAM GNCS-2022 InSANE (CUP_E55F22000270001).
[*]Corresponding author.
stefania.costantini@univaq.it (S. Costantini); andrea.formisano@uniud.it (A. Formisano)
0000-0002-5686-6124 (S. Costantini); 0000-0002-6755-9314 (A. Formisano)

Recent work summarized in [11] has been aimed at extending to Epistemic Logic Programming some notions which have been previously defined for ASP, where many useful results have stemmed from them. So, according to [11, 12, 13], analogous properties might prove useful in ELPs as well. In particular, they consider *splitting* (introduced for ASP in [14]), which allows a program to be (iteratively) divided into parts ("top" and "bottom") in a principled way: the answer sets of a given program can be computed incrementally, starting from the answer sets of the bottom, which are used to simplify the top, and then the union of each answer set of the bottom with each answer set of the corresponding simplified top forms an answer set of the overall program. They extend to ELPs the concept of splitting and the method of incremental calculation of the semantics (here, it is the world views that must be calculated). This by defining a notion of *Epistemic Splitting*, where top and bottom are defined w.r.t. the occurrence of epistemic operators. Further, they adapt to ELPs other properties of ASP, which are implied by this property, namely the fact that adding constraints leads to reduce the number of answer sets, for ELPs, according to them, of the world views (*Subjective Constraint Monotonicity*), and *Foundedness*, meaning that atoms composing answer sets cannot have been derived through cyclic positive dependencies (where, for ELPs, they redefine positive dependencies so as to involve epistemic operators). In substance, this approach establishes properties that a semantics should fulfil, and then they compare the existing semantics with respect to these properties.

In this paper, we explore a different stance: in order to establish a term of comparison among the various semantics, we introduce a semantic approach which is very plainly based on the basic understanding of ELP and world views. We then experiment with the new approach on many examples taken from the relevant literature, and we "observe" its behaviour, in terms of the correspondence or discrepancy with the results returned by other relevant semantic approaches. The paper is organized as follows. In Sections 2 and 3 we recall ASP and ELPs. In Section 4 we introduce and discuss, via many examples, our proposal. Finally, in Section 5 we conclude.

2. Answer Set Programming and Answer Set Semantics

In ASP, one can see an answer set program (for short 'ASP program') as a set of statements that specify a problem, where each answer set represents a solution compatible with this specification. Whenever an ASP program has no answer sets (no solution can be found), it is said to be *inconsistent*, otherwise it is said to be *consistent*. Several well-developed freely available *answer set solvers* exist that compute the answer sets of a given program. Syntactically, an ASP program Π is a collection of *rules* of the form

$$A_1| \dots |A_g \leftarrow\ L_1, \dots, L_n.$$

where each A_i, $0 \leq i \leq g$, is an atom and $|$ indicates disjunction, and the L_is, $0 \leq i \leq n$, are literals (i.e., atoms or negated atoms of the form $not\ A$). The left-hand side and the right-hand side of the rule are called *head* and *body*, resp. A rule with empty body is called a *fact*. Notation $A \,|\, B$ indicates disjunction, usable only in rule heads and, so, in facts. A rule with empty head (or, equivalently, with head $\bot$), of the form '$\leftarrow L_1, ..., L_n.$' or '$\bot \leftarrow L_1, ..., L_n.$', is a *constraint*, stating that literals $L_1, \dots, L_n$ are not allowed to be simultaneously true in any answer set; the impossibility to fulfil such requirement is one of the reasons that make a program inconsistent.

All extensions of ASP not explicitly mentioned above are not considered in this paper. We implicitly refer to the “ground” version of Π, which is obtained by replacing in all possible ways the variables occurring in Π with the constants occurring in Π itself, and it is thus composed of ground atoms, i.e., atoms which contain no variables.

The answer set (or “stable model”) semantics can be defined in several ways [15, 16]. However, answer sets of a program Π, if any exists, are the supported minimal classical models of the program interpreted as a first-order theory in the obvious way. The original definition from [3], introduced for programs where rule heads were limited to be single atoms, was in terms of the ‘GL-Operator’. Given set of atoms I and program Π, $GL_{\Pi}(I)$ is defined as the least Herbrand model of the program Π^I, namely, the (so-called) Gelfond-Lifschitz reduct of Π w.r.t. I. Π^I is obtained from Π by: 1. removing all rules which contain a negative literal $not\, A$ such that $A \in I$; and 2. removing all negative literals from the remaining rules. The fact that Π^I is a positive program ensures that a least Herbrand model exists and can be computed via the standard immediate consequence operator [17]. Then, I is an answer set whenever $GL_{\Pi}(I) = I$.

3. Epistemic Logic Programs and Their Properties

Epistemic Logic Programs allow one to express within ASP programs so-called *subjective literals* (in addition to *objective literals*, that are those that can occur in plain ASP programs, plus the truth constants ⊤ and ⊥). Such new literals are constructed via the *epistemic operator* **K** (disregarding without loss of generality the other epistemic operators). The literal $\mathbf{K}A$ means that the (ground) atom A is true in every answer set of given program Π (it is a *cautious consequence* of Π). The syntax of rules is analogous to ASP, save that literals in the body of rules now can be either objective or subjective. Nesting of subjective literals is not considered here. An ELP program is called *objective* if no subjective literals occur therein, i.e., it is an ASP program. A constraint involving (also) subjective literals is called a *subjective constraint*, where one involving objective literals only is an *objective constraint*. Let At be the set of atoms occurring (within either objective or subjective literals) in a given program Π, and $Atoms(r)$ be the set of atoms occurring in rule r. By some abuse of notation, we denote by $Atoms(X)$ the set of atoms occurring in X, whatever X is (a rule, a program, an expression, etc.). Let $Head(r)$ be the head of rule r and $Body_{obj}(r)$ (resp., $Body_{subj}(r)$) be the (possibly empty) set of objective (resp., subjective) literals occurring in the body of r. For simplicity, we often write $Head(r)$ and $Body_{obj}(r)$ in place of $Atoms(Head(r))$ and $Atoms(Body_{obj}(r))$, respectively, when the intended meaning is clear from the context. We call *subjective rules* those rules whose body is made of subjective literals only.

The semantics of ELPs is based on the notion of *world views*: namely, sets of answer sets. Each world view determines the truth value of all objective literals in a program. For example, the program $\{a \leftarrow not\, b,\ b \leftarrow not\, a,\ e \leftarrow not\, \mathbf{K}f,\ f \leftarrow not\, \mathbf{K}e\}$, under every semantics, has two world views: $[\{a, e\}, \{b, e\}]$, where $\mathbf{K}e$ is true and $\mathbf{K}f$ is false, and $[\{a, f\}, \{b, f\}]$ where $\mathbf{K}f$ is true and $\mathbf{K}e$ is false. Note that, according to a widely-used convention, each world view, which is a set of answer sets, is enclosed in square brackets []. The presence of two answer sets in each world view is due to the cycle on objective atoms, whereas the presence of two world views is due to the cycle on subjective atoms (in general, the existence and the number of world views is related to such cycles, cf., [18] for a detailed discussion).

Let a semantics $\mathcal{S}$ be a function mapping each program into sets of 'belief views', i.e., sets of sets of objective literals, where $\mathcal{S}$ has the property that, if Π is an objective program, then the unique member of $\mathcal{S}(\Pi)$ is the set of stable models of Π. Given a program Π, each member of $\mathcal{S}(\Pi)$ is called an $\mathcal{S}$-*world view* of Π (we will often write "world view" in place of "$\mathcal{S}$-world view" whenever mentioning the specific semantics is irrelevant). As usual, for any world view W and any subjective literal $\mathbf{K}L$, we write $W \models \mathbf{K}L$ iff for all $I \in W$ the literal L is satisfied by I (i.e., if $L \in I$ for L atom, or $A \notin I$ if L is $not\ A$). W satisfies a rule r if each $I \in W$ satisfies r.

The property of *Subjective Constraint Monotonicity* states that, for any epistemic program Π and any subjective constraint r, W is a world view of $\Pi \cup \{r\}$ iff both W is a world view of Π and W satisfies r. Thus, if this property is fulfilled by a semantic $\mathcal{S}$, a constraint can rule out world views but cannot rule out some answer set from within a world view (or, equivalently, cannot substitute a world view with a new one). We report below some of the most relevant semantic definitions for ELPs. We start with the seminal definition of the first ELP semantics, introduced in [2], that we call for short G94. Let Π be an ELP program, and r a rule occurring therein.

Definition 3.1 (G94-world views). *The G94-reduct of* Π *with respect to a non-empty set of interpretations W is obtained by: (i) replacing by $\top$ every subjective literal $L \in Body_{subj}(r)$ such that L is of the form* $\boldsymbol{K}L$ *and $W \models L$, and (ii) replacing all other occurrences of subjective literals of the form* $\boldsymbol{K}L$ *by $\bot$. A non-empty set of interpretations W is a G94-world view of* Π *iff W coincides with the set of all stable models of the G94-reduct of* Π *with respect to W.*

This definition was then extended to a new one [4], that we call for short G11.

Definition 3.2 (G11-world views). *The G11-reduct of* Π *with respect to a non-empty set of interpretations W is obtained by: (i) replacing by $\bot$ every subjective literal $L \in Body_{subj}(r)$ such that $W \not\models L$, (ii) removing all other occurrences of subjective literals of the form $\neg\boldsymbol{K}L$. (iii) replacing all other occurrences of subjective literals of the form $\boldsymbol{K}L$ by L. A non-empty set of interpretations W is a G11-world view of* Π *iff W coincides with the set of all stable models of the G11-reduct of* Π *w.r.t. W.*

Notice that, $\neg\mathbf{K}L$ is usually indicated as $not\ \mathbf{K}L$ in examples. In [11], it is noticed that K15 [19], reported below, slightly generalizes the semantics proposed in [4].

Definition 3.3 (K15-world views). *The K15-reduct of* Π *with respect to a non-empty set of interpretations W is obtained by: (i) replacing by $\bot$ every subjective literal $L \in Body_{subj}(r)$ such that $W \not\models L$, and (ii) replacing all other occurrences of subjective literals of the form $\boldsymbol{K}L$ by L. A non-empty set of interpretations W is a K15-world view of* Π *iff W coincides with the set of all stable models of the K15-reduct of* Π *w.r.t. W.*

Semantics G11 and K15, that are refinements of the original G94 semantics, have been proposed over time to cope with new examples that were discovered, on which existing semantic approaches produced unwanted or unintuitive world views.

K15 can be seen as a basis for the semantics proposed in [7] (called S16 for short). In particular, S16 treats K15 world views as candidate solutions, to be pruned in a second step, where some world views are removed, by applying the principle of keeping those which maximize what is not known. World views in [7] are obtained in particular as follows, where note however that they consider the operator **not**, that can be rephrased as $not\,\mathbf{K}A$ where not is ASP standard 'default negation' (meaning that A must be false in some answer set of a given world view).

Definition 3.4 (S16-world views). *Let $EP(\Pi)$ be the set of literals of the form **not** F occurring in given program Π. Given $\Phi \subseteq EP(\Pi)$, the* Epistemic reduct Π^{Φ} *of Π w.r.t. Φ is obtained by: (i) replacing every **not** $F \in \Phi$ with true, and (ii) replacing every **not** $F \notin \Phi$ with $not\,F$. Then, the set $\mathcal{A}$ of the answer sets of Π^{Φ} is a* candidate world view *if every **not** $F \in \Phi$ is true w.r.t. $\mathcal{A}$ (i.e., F is false in some answer set $J \in \mathcal{A}$) and every **not** $F \notin \Phi$ is false (i.e., F is true in every answer set $J \in \mathcal{A}$). We say that $\mathcal{A}$ is obtained from Φ, or is corresponding to Φ, or that it is a candidate world view w.r.t. Φ, where Φ is called a* candidate valid guess. *Then, $\mathcal{A}$ is an S16* world view *if it is maximal, i.e., if there exists no other candidate world view obtained from guess Φ' where $\Phi \subset \Phi'$ (so, Φ is called a* valid guess*).*

All the above semantics, in order to check whether a belief view $\mathcal{A}$ is indeed a world view, adopt some kind of reduct, reminiscent of that related to the stable model semantics, and $\mathcal{A}$ is a world view if it is *stable* w.r.t. this reduct. The F15 semantics [6, 20] is based on very different principles, namely, it is based on a combination of Equilibrium Logic [21, 22, 23] with the modal logic S5. There, an EHT interpretation associates, via a function h, a belief view $\mathcal{A}$ with another belief view $\mathcal{A}'$ composed, for every set $A \in \mathcal{A}$, of sets $A' \subseteq A$. The purpose is to state that an implication is entailed, in any "belief point", i.e., in any interpretation $A \in \mathcal{A}$, by the couple $\langle \mathcal{A}, \mathcal{A}' \rangle$ if it is entailed either by $\mathcal{A}$ or by $\mathcal{A}'$. An EHT interpretation satisfies a theory in the usual way, and is total on a subset $\mathcal{X}$ of $\mathcal{A}$ if h gives back sets in $\mathcal{X}$ unchanged. A *total EHT model* can be an *equilibrium EHT model*, and is defined to be an **F15 world view**, if it is minimal according to two particular minimality conditions (not reported here).

Differently from F15, FAAEL [13] is based on the modal logic KD45. To define FAAEL, a belief view is transformed from a set of interpretations to a set of HT-interpretations, i.e., interpretations in terms of the logic of Here-and-There (HT) [24] which are couples $\langle H, T \rangle$ of 'plain' interpretations. A belief view is *total* if $H = T$ for all composing interpretations, thus reducing to the previous notion of belief view. A total version of any belief view can be formed, taking all the T's as components. A belief interpretation is now a belief view plus an HT interpretation, say $\hat{H}$, possibly not belonging to the belief view. The peculiarity of the entailment relation (defined in terms of HT logic) is in the implication, that must hold (in the usual way) in the belief interpretation, but also in the total version of the belief view therein. For total belief interpretations, the new relation collapses to the modal logic KD45. An epistemic interpretation is defined to be a belief model if all its composing HT interpretation as well as $\hat{H}$ entail all formulas of given theory. It is an epistemic model, if $\hat{H}$ is among the composing interpretations, and it is an *equilibrium belief model* if it satisfies certain minimality conditions. A belief view is a **FAAEL world view** if it is "extracted" from an equilibrium belief model $\mathcal{E}$ by taking all the T components of each $\langle H, T \rangle$ which is found in $\mathcal{E}$.

For formal definitions of F15 and FAAEL, that for lack of space we cannot report here, we refer the reader to the aforementioned references. FAAEL satisfies [12] Epistemic Splitting, Subjective Constraint Monotonicity, and Foundedness. G94 satisfies Epistemic Splitting, Subjective Constraint Monotonicity, but not Foundedness. In [13], it is proved that FAAEL world views coincide

program	world views
$a \vee b$	$[\{a\},\{b\}]$
$a \vee b$ $a \leftarrow \mathbf{K}b$	$[\{a\},\{b\}]$
$a \vee b$ $a \leftarrow not\,\mathbf{K}b$	$[\{a\}]$
$a \vee b$ $c \leftarrow not\,\mathbf{K}b$	$[\{a,c\},\{b,c\}]$
$a \leftarrow not\,\mathbf{K}b$ $b \leftarrow not\,\mathbf{K}a$	$[\{a\}],\ [\{b\}]$
$a \leftarrow not\,\mathbf{K}not\,a$ $a \leftarrow not\,\mathbf{K}a$	$[\{a\}]$

program	G94/G11/FAEEL	K15/F15/S16
$a \leftarrow not\,\mathbf{K}not\,a$	$[\emptyset],\ [\{a\}]$	$[\{a\}]$
$a \vee b$ $a \leftarrow not\,\mathbf{K}not\,b$	none	$[\{a\}]$
$a \vee b$ $a \leftarrow \mathbf{K}not\,b$	$[\{a\}],\ [\{a\},\{b\}]$	$[\{a\},\{b\}]$
$a \leftarrow b$ $b \leftarrow not\,\mathbf{K}not\,a$	$[\emptyset],\ [\{a,b\}]$	$[\{a,b\}]$
$a \leftarrow not\,\mathbf{K}not\,b$ $b \leftarrow not\,\mathbf{K}not\,a$	$[\emptyset],\ [\{a\},\{b\}]$	$[\{a\},\{b\}]$

Figure 1: On the left, examples where G94, G11, K15, F15, S16, and FAEEL agree. On the right, examples where G94/G11/FAEEL differ from K15/F15/S16. (Figure taken from [13].)

program	G94	G11/FAEEL	K15	F15/S16
$a \leftarrow not\,\mathbf{K}not\,b\ \wedge\ not\,b$ $b \leftarrow not\,\mathbf{K}not\,a\ \wedge\ not\,a$	$[\emptyset],[\{a\},\{b\}]$			$[\{a\},\{b\}]$
$a \leftarrow \mathbf{K}a$	$[\emptyset],\ [\{a\}]$	$[\emptyset]$		
$a \leftarrow \mathbf{K}a$ $a \leftarrow not\,\mathbf{K}a$	$[\{a\}]$	none		

Figure 2: Examples showing differences among several semantics. (Figure taken from [13].)

with *founded* G94 world views, where (roughly) founded world views are those where in every composing interpretation, objective atom G is never derived, directly or indirectly, from $\mathbf{K}G$.

We apologize with the readers and with the authors, because, for lack of space, we do not consider other recent semantics, such as [25, 9, 26].

In Figures 1 and 2 a summary is reported, taken from [13], of how the semantics presented above behave on some examples which are considered to be significant of situations that can be found in practical programming.

4. Our Observations and Proposal

We expose the new method, and we experiment it, taking as a base the examples proposed in Figures 1 and 2, with few others.

Let us notice that, actually, in Gelfond's intuition, $\mathbf{K}G$ means that G is true in all the answer set of a given program, where the set of these answer sets is now called world view, or that G is true in all the answer sets of a certain world view, if there are many of them. It is not really required for G to be derivable from the program in a 'founded' way as it happens in ASP, or, at least, the concept of founded derivation becomes different.

In the GL94 computation of a world view, what is assumed to be known or not known comes from the world view, not from the program. What is required by this basic approach is that a world view is consistent w.r.t. the program, in the sense that what is assumed to be known is indeed concluded, and what is assumed to be false is not concluded. However, the point is that subjective atoms appearing in the program (and that are not derived, but elicited from the underlying world view) have a role in drawing conclusions.

We introduce an approach where this seminal intuition is literally applied. We then put the new approach to work on a number of examples, taking the occasion for a comparison with the semantics we have introduced above.

4.1. A New Approach

We consider in this context only subjective literals $\mathbf{K}G$ and $\mathbf{K}\neg G$, the latter with notation $\mathbf{K}not\ G$. We will consider them as new atoms, called *knowledge atoms*. Negation not in front of knowledge atoms is assumed to be the standard default negation. So, instead of ELPs proper, we here consider (equivalently) ASP programs possibly involving knowledge atoms.

First of all we introduce the concept of internal consistency of a set of atoms including knowledge atoms.

Definition 4.1. *A set A of atoms, composed of objective atoms and knowledge atoms, is said to be* knowledge consistent *iff:*

(i) it contains G whenever it contains $\boldsymbol{K}G$;
(ii) it does not contain G whenever it contains $\boldsymbol{K}not\ G$.

A set of sets of atoms $\mathcal{W}$, each such set composed of objective atoms and knowledge atoms, is called here *epistemic interpretation*.

Definition 4.2. *Given ASP program P possibly involving knowledge atoms, let $SMC(P)$ be the set of those answer sets of the program which are knowledge-consistent.*

Property 1. *$SMC(P)$ correspond to the stable models of the program P' obtained from P by adding, for each special atom $\boldsymbol{K}G$ or $\boldsymbol{K}not\ G$ occurring in P, constraints:*

$$\leftarrow \boldsymbol{K}G, not\ G$$
$$\leftarrow \boldsymbol{K}not\ G, G$$

To the aim of establishing a uniform comparison among various semantic approaches, we propose a basic point of view on ELPs, that for convenience we present as a new semantics.

Definition 4.3. *[CF22-adaptation] The CF22-adaptation $\Gamma - \mathcal{W}$ of a program Γ with respect to an epistemic interpretation $\mathcal{W}$ is obtained by adding to Γ:*

(i) new fact $\boldsymbol{K}G$ whenever $\mathcal{W} \models G$, and
(ii) new fact $\boldsymbol{K}not\ G$ whenever $\mathcal{W} \models \neg G$.

Let $F_{\mathcal{W}}$ be the set of those newly added facts of the form $\boldsymbol{K}G$.

Definition 4.4 (CF22 world view). *An epistemic interpretation $\mathcal{W}$ is called a CF22 world view of a theory Γ if $\mathcal{W} = SM'(\Gamma - \mathcal{W})$, where $SM'(\Gamma - \mathcal{W})$ is obtained from $SMC(\Gamma - \mathcal{W})$ by removing all knowledge atoms.*

As seen, the S16 semantics maximizes what is not known, which is equivalent to minimizing what is known. The proposers of S16 consider each potential world view (that in their approach is associated to a guess about what is not known) as a candidate world view, and discard those for which there exists another one with a larger guess on what is not known (equivalently, a smaller guess on what is known), in terms of set inclusion. Rephrasing their criterion in terms of our approach, we have:

Definition 4.5 (S16 Criterion - CF22+S16C). *Each world view $\mathcal{W}$ as of Def. 4.4 is considered to be a* candidate world view. *A candidate world view $\mathcal{W}$ is indeed a world view under CF22+S16C if no other candidate world view $\mathcal{W}$' exists, where $F_{\mathcal{W}'} \subset F_{\mathcal{W}}$.*

4.2. CF22 World Views: Examples of Application

It can be easily seen that, on the examples on the left-column table of Fig. 1, on which all the above-presented semantic approaches agree, CF22 agrees as well. Below we present in detail a number of examples, some taken from Figures 1 (right-column) and 2, and some from the relevant literature. The aim is to employ CF22 as a term of comparison among the various semantics.

4.2.1. Example 1

Consider the program Γ composed of a single rule

$a \leftarrow not\, \mathbf{K}a.$

and the epistemic interpretation $\mathcal{W} = [\emptyset]$. According to Def. 4.3, the added fact is:

$\mathbf{K}not\, a.$

We have that $SMC(\mathit{\Gamma} - \mathcal{W}) = \emptyset$, thus $SM'(\mathit{\Gamma} - \mathcal{W}) = \emptyset$, so $\mathcal{W}$ **is not** a CF22 world view. To see how this has been obtained, notice that $\mathit{\Gamma} - \mathcal{W}$ has the unique answer set $\{\mathbf{K}not\, a, a\}$ where $\mathbf{K}not\, a$ is a fact, and a is derived from $not\, \mathbf{K}a$, via default negation as fact $\mathbf{K}a$ is not present; this answer set is however not knowledge consistent, as it contains a where it says that a is not known; thus, the set of knowledge consistent answer sets of $\mathit{\Gamma} - \mathcal{W}$ is empty.

Consider now the epistemic interpretation $\mathcal{W} = [\{a\}]$. According to Def. 4.3, one fact is added:

$\mathbf{K}a.$

We have that $SMC(\mathit{\Gamma} - \mathcal{W}) = [\{\mathbf{K}a\}]$, thus $SM'(\mathit{\Gamma} - \mathcal{W}) = [\emptyset]$, so $\mathcal{W}$ **is not** a CF22 world view. Therefore, in accordance to all other semantics, this program has no CF22 world views.

4.2.2. Example 2: Cyclic Dependence

Consider the program Γ composed of the two rules

$a \leftarrow not\, \mathbf{K}b.$

$b \leftarrow not\, \mathbf{K}a.$

and the epistemic interpretation $\mathcal{W} = [\emptyset]$. According to Def. 4.3, the added facts are:

$\mathbf{K}not\, a. \qquad \mathbf{K}not\, b.$

We have that $SMC(\mathit{\Gamma} - \mathcal{W}) = \emptyset$, thus $SM'(\mathit{\Gamma} - \mathcal{W}) = \emptyset$, so $\mathcal{W}$ **is not** a CF22 world view. To see how this has been obtained, notice that $\mathit{\Gamma} - \mathcal{W}$ has the unique answer set $\{\mathbf{K}not\, a, \mathbf{K}not\, b, a, b\}$ where $\mathbf{K}not\, a, \mathbf{K}not\, b$ are facts, a is derived from $not\, \mathbf{K}b$, as fact $\mathbf{K}b$ is not present, and similarly for b; this answer set is however not knowledge consistent, as it contains a and b where it says that they are not known, thus the set of knowledge consistent answer sets of $\mathit{\Gamma} - \mathcal{W}$ is empty.

Consider now the epistemic interpretation $\mathcal{W} = [\{a\}]$ (the analogous can be done for $[\{b\}]$). According to Def. 4.3, the added facts are:

$\mathbf{K}a.$ $\quad$ $\mathbf{K}\mathit{not}\, b.$

We have that $SMC(\mathit{\Gamma} - \mathcal{W}) = [\{Ka, Knot\, b, a\}]$, thus $SM'(\mathit{\Gamma} - \mathcal{W}) = [\{a\}]$, so $\mathcal{W}$ **is** a CF22 world view (analogously for $[\{b\}]$).

Consider the epistemic interpretation $\mathcal{W} = [\{a, b\}]$. In this case the added facts are:

$\mathbf{K}a.$ $\quad$ $\mathbf{K}b.$

We have that $SMC(\mathit{\Gamma} - \mathcal{W}) = [\{Ka, Kb\}]$, thus $SM'(\mathit{\Gamma} - \mathcal{W}) = [\emptyset]$, so $\mathcal{W}$ **is not** a CF22 world view.

Finally, for the epistemic interpretation $\mathcal{W} = [\{a\}, \{b\}]$, according to Def. 4.3, there are no added facts. We have that $SMC(\mathit{\Gamma} - \mathcal{W}) = [\{a, b\}]$ (each atom a, b derived from not knowing the other), thus $SM'(\mathit{\Gamma} - \mathcal{W}) = [\{a, b\}]$, so $\mathcal{W}$ **is not** a CF22 world view.

Also on this example, CF22 agrees with all other semantics.

4.2.3. Example 3

Consider the program Γ

$a \vee b.$

$a \leftarrow \mathbf{K}b.$

$b \leftarrow \mathbf{K}a.$

and the epistemic interpretation $\mathcal{W} = [\emptyset]$. According to Def. 4.3, the added facts are:

$\mathbf{K}\mathit{not}\, a.$ $\quad$ $\mathbf{K}\mathit{not}\, b.$

We have that the two rules cannot be applied, and the disjunction would generate answer sets $\{a\}$ and $\{b\}$ that are not knowledge consistent; thus, $SM'(\mathit{\Gamma} - \mathcal{W}) = \emptyset$, so $\mathcal{W}$ **is not** a CF22 world view.

Consider the epistemic interpretation $\mathcal{W} = [\{a\}]$ (the analogous can be done for $[\{b\}]$). According to Def. 4.3, the added facts are:

$\mathbf{K}a.$ $\quad$ $\mathbf{K}\mathit{not}\, b.$

We have the answer set $\{\mathbf{K}a, \mathbf{K}\mathit{not}\, b, b\}$ where b is derived from the second rule. However, this answer set is not knowledge consistent; thus, $SM'(\mathit{\Gamma} - \mathcal{W}) = \emptyset$, so $\mathcal{W}$ **is not** a CF22 world view.

Consider $\mathcal{W} = [\{a, b\}]$. According to Def. 4.3, the added facts are:

$\mathbf{K}a.$ $\quad$ $\mathbf{K}b.$

We have that $SMC(\mathit{\Gamma} - \mathcal{W}) = [\{\mathbf{K}a, \mathbf{K}b, a, b\}]$, with atoms a and b derived via the rules given the facts; this answer set is knowledge consistent, thus $SM'(\mathit{\Gamma} - \mathcal{W}) = [\{a, b\}]$, so $\mathcal{W}$ **is** a CF22 world view.

Consider finally $\mathcal{W} = [\{a\}, \{b\}]$. According to Def. 4.3, there are no added facts. We have that $SMC(\mathit{\Gamma} - \mathcal{W}) = SM'(\mathit{\Gamma} - \mathcal{W}) = [\{a\}, \{b\}]$, deriving from the disjunction, as the two rules cannot be applied; thus, $\mathcal{W}$ **is** a CF22 world view.

This example shows that CF22, that here agrees with G11, does not satisfy foundedness. However, if one augments it with the S16 Criterion (we call the combination CF22+S16C) then the unfounded world view $[\{a, b\}]$ is excluded, as there exists $SM'(\mathit{\Gamma} - \mathcal{W}) = [\{a\}, \{b\}]$ which is based on fewer added positive knowledge literals (none for the latter and $\mathbf{K}a$ and $\mathbf{K}b$ for the former).

One may notice that, for world view $[\{a, b\}]$, these atoms are not derived from the program via a positive circularity: rather, they are supported, in the program, from what is deemed to be known in the world view itself. So, while this world view can be excluded by applying a minimality criterion, it is however not unreasonable in itself.

It can be seen that simpler example

$a \leftarrow \mathbf{K}b.$

$b \leftarrow \mathbf{K}a.$

has CF22 world views $[\emptyset]$ and $[\{a, b\}]$, where the latter would be discarded under CF22+S16C.

4.2.4. Example 4

Consider the program Γ:

$a \leftarrow \mathit{not}\,\mathbf{K}\mathit{not}\,a.$

and the epistemic interpretation $\mathcal{W} = [\emptyset]$. According to Def. 4.3, the added fact is:

$\mathbf{K}\mathit{not}\,a.$

We have that $SMC(\Gamma - \mathcal{W}) = [\{\mathbf{K}\mathit{not}\,a\}]$, thus $SM'(\Gamma - \mathcal{W}) = [\emptyset]$, so $\mathcal{W}$ **is** a CF22 world view.

Consider the epistemic interpretation $\mathcal{W} = [\{a\}]$. According to Def. 4.3, the added fact is:

$\mathbf{K}a.$

We have that $SMC(\Gamma - \mathcal{W}) = [\{\mathbf{K}a, a\}]$ (as fact $\mathbf{K}\mathit{not}\,a$ is not present, its negation is true), thus $SM'(\Gamma - \mathcal{W}) = [\{a\}]$, so $\mathcal{W}$ **is** a CF22 world view.

On this example, CF22 agrees with G94, G11, FAAEL.

4.2.5. Example 5

Let us now consider a more problematic example:

$a \leftarrow \mathbf{K}a.$

$a \leftarrow \mathit{not}\,\mathbf{K}a.$

and the epistemic interpretation $\mathcal{W} = [\emptyset]$. According to Def. 4.3, the added fact is:

$\mathbf{K}\mathit{not}\,a.$

We have that $SMC(\Gamma - \mathcal{W}) = \emptyset$ (as fact $\mathbf{K}a$ is not present, its negation is true, thus allowing to derive a, within however a stable model which is not knowledge consistent), thus $SM'(\Gamma - \mathcal{W}) = \emptyset \neq [\emptyset]$, so $\mathcal{W}$ **is not** a CF22 world view.

Consider the epistemic interpretation $\mathcal{W} = [\{a\}]$. The added fact is:

$\mathbf{K}a.$

We have that $SMC(\Gamma - \mathcal{W}) = [\{\mathbf{K}a, a\}]$, thus $SM'(\Gamma - \mathcal{W}) = [\{a\}]$, so $\mathcal{W}$ **is** a CF22 world view.

On this example, CF22 agrees with G94, where however all the other semantics provide no world view.

If the program would simply be

$a \leftarrow \mathbf{K}a.$

then its world views, as can be easily seen, would be $[\emptyset]$ and $[\{a\}]$.

On this example, CF22 agrees with G94. It would agree with G11, K15, F15, S16, FAAEL under CF22+S16C.

4.2.6. Example 6

In previous examples, CF22+S16C tended to agree with S16. This is however not always the case. Given the rules:

$a \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, b, \mathit{not}\, b.$

$b \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, a, \mathit{not}\, a.$

consider the epistemic interpretation $\mathcal{W} = [\emptyset]$. According to Def. 4.3, the added facts are:

$\mathbf{K}\mathit{not}\, a. \qquad \mathbf{K}\mathit{not}\, b.$

We have that $\mathit{SMC}(\mathit{\Gamma} - \mathcal{W}) = [\mathbf{K}\mathit{not}\, a, \mathbf{K}\mathit{not}\, b]$, thus $\mathit{SM}'(\mathit{\Gamma} - \mathcal{W}) = [\emptyset]$, so $\mathcal{W}$ **is** a CF22 world view.

Consider the epistemic interpretation $\mathcal{W} = [\{a\}]$ (one can proceed analogously for $[\{b\}]$). According to Def. 4.3, the added facts are:

$\mathbf{K}a. \qquad \mathbf{K}\mathit{not}\, b.$

We have that $\mathit{SMC}(\mathit{\Gamma} - \mathcal{W}) = \emptyset$ (as one can derive b, obtaining however a stable model which is not knowledge consistent, because of fact $\mathbf{K}\mathit{not}\, b$), thus $\mathit{SM}'(\mathit{\Gamma} - \mathcal{W}) = \emptyset$, so $\mathcal{W}$ **is not** a CF22 world view.

On this example, CF22 agrees with G94, where however all the other semantics provide no world view.

Consider the epistemic interpretation $\mathcal{W} = [\{a\}, \{b\}]$, where there are no added facts. We have that $\mathit{SMC}(\mathit{\Gamma} - \mathcal{W}) = \mathit{SM}'(\mathit{\Gamma} - \mathcal{W}) = [\{a\}, \{b\}]$, so $\mathcal{W}$ **is** a CF22 world view. Epistemic interpretation $[\{a, b\}]$ is easily discarded.

On this example, S22 agrees with G94, G11, K15, FAAEL. Under CF22+S16C nothing changes, as both CF22 world views do not rely on positive knowledge atoms.

If the program is (seemingly) simpler, i.e.:

$a \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, b.$

$b \leftarrow \mathit{not}\, \mathbf{K}\mathit{not}\, a.$

we have that, similarly to before, $\{a\}$ and $\{b\}$ are not CF22 world views. However, $\mathit{SMC}(\mathit{\Gamma} - \mathcal{W}) = \emptyset$ now is a CF22 world view, because from added facts

$\mathbf{K}\mathit{not}\, a. \qquad \mathbf{K}\mathit{not}\, b.$

one does not derive anything. Instead, $\mathcal{W} = [\{a\}, \{b\}]$ is not, because with no added facts one can derive both a and b, so $\mathit{SMC}(\mathit{\Gamma} - \mathcal{W}) = \mathit{SM}'(\mathit{\Gamma} - \mathcal{W}) = [\{a, b\}]$.

But, $\mathcal{W} = [\{a, b\}]$ is a CF22, world view, because adding new facts

$\mathbf{K}a. \qquad \mathbf{K}b.$

both negations in the bodies of the program's two rules are true, so one derives both a and b obtaining $\mathit{SMC}(\mathit{\Gamma} - \mathcal{W}) = \mathit{SM}'(\mathit{\Gamma} - \mathcal{W}) = [\{a, b\}]$.

This program, under CF22, has the world views $[\emptyset]$ and $[\{a, b\}]$. The rationale underlying world view $[\{a, b\}]$ is that, again, it is consistent with the given program, relatively to the positive knowledge atoms that the world view entails.

4.2.7. Example 7

Consider the epistemic logic program:

$a \vee b.$

$a \leftarrow \mathbf{K} \mathit{not}\, b.$

Clearly, because of the disjunction $[\emptyset]$ cannot be a CF22 world view. Consider the epistemic interpretation $\mathcal{W} = [\{a\}]$ According to Def. 4.3, the added facts are:

$\mathbf{K}a. \qquad \mathbf{K}\mathit{not}\, b.$

We have that $SMC(\Gamma - \mathcal{W}) = [\{\mathbf{K}a, \mathbf{K}\mathit{not}\, b, a\}]$, thus $SM'(\Gamma - \mathcal{W}) = [\{a\}]$, so $\mathcal{W}$ **is** a CF22 world view.

Consider the epistemic interpretation $\mathcal{W} = [\{b\}]$. Now, the added facts are:

$\mathbf{K}b. \qquad \mathbf{K}\mathit{not}\, a.$

We have that $SMC(\Gamma - \mathcal{W}) = [\{\mathbf{K}b, \mathbf{K}\mathit{not}\, a, b\}]$, thus $SM'(\Gamma - \mathcal{W}) = [\{b\}]$, so $\mathcal{W}$ **is** a CF22 world view.

Consider the epistemic interpretation $\mathcal{W} = [\{a\}\{b\}]$. According to Def. 4.3, there are no added facts. We have that $SMC(\Gamma - \mathcal{W}) = SM'(\Gamma - \mathcal{W}) = [\{a\}, \{b\}]$, so $\mathcal{W}$ **is** a CF22 world view.

It is easy to verify that instead $[\{a, b\}]$ is not a CF22 world view (because the disjunction cannot generate both a and b).

On this example, CF22 does not agree with existing semantics, because of the world view $[\{b\}]$, that they do not produce. Under CF22+S16C, there is agreement with S16, as in fact world view $[\{a\}, \{b\}]$, based upon an empty set of added knowledge atoms of the form $\mathbf{K}A$, rules out both $[\{a\}]$ and $[\{b\}]$.

4.2.8. Example 8

Consider the program:

$a \vee b.$

$\leftarrow \mathit{not}\, \mathbf{K}a.$

Clearly, because of the disjunction $[\emptyset]$ cannot be a CF22 world view. Consider the epistemic interpretation $\mathcal{W} = [\{a\}]$ According to Def. 4.3, the added facts are:

$\mathbf{K}a. \qquad \mathbf{K}\mathit{not}\, b.$

We have that $SMC(\Gamma - \mathcal{W}) = [\{\mathbf{K}a, \mathbf{K}\mathit{not}\, b, a\}]$ (the stable model with b is excluded as it is not knowledge consistent), thus $SM'(\Gamma - \mathcal{W}) = [\{a\}]$, so $\mathcal{W}$ **is** a CF22 world view.

Consider the epistemic interpretation $\mathcal{W} = [\{b\}]$. According to Def. 4.3, the added facts are:

$\mathbf{K}b. \qquad \mathbf{K}\mathit{not}\, a.$

The constraint is clearly violated, then we have $SMC(\Gamma - \mathcal{W}) = SM'(\Gamma - \mathcal{W}) = \emptyset$, thus $\mathcal{W}$ **is not** a CF22 world view.

Consider now $\mathcal{W} = [\{a\}\{b\}]$. According to Def. 4.3, there are no added facts. The constraint is violated, then we have $SMC(\Gamma - \mathcal{W}) = SM'(\Gamma - \mathcal{W}) = \emptyset$, thus $\mathcal{W}$ **is not** a CF22 world view. It is easy to verify that also $[\{a, b\}]$ is not a CF22 world view (because the constraint is satisfied, but the disjunction cannot generate both a and b). Thus, CF22 on this program agrees with K15 and S16, and, like them, it does not satisfy *Subjective Constraint Monotonicity* as defined in [10] and subsequent papers. This property imposes that a constraint, in the above example

$\leftarrow \mathit{not}\, \mathbf{K}a.$

put at a higher level (in the sense of Lifschitz and Turner splitting notion, extended in the above-mentioned works to ELPs) w.r.t. an “object program”, that in the above example is

$a \vee b.$

might have one of the following two effects: (i) the constraint is respected in a world view of the object (or "bottom"), program, thus such world view remains untouched; or, (ii) the constraint is violated in a world view, and in this case the world view is excluded. In particular, according to the FAAEL semantics, that satisfies Subjective Constraint Monotonicity, the above program has no world views, since the unique world view of the bottom part, i.e., $[\{a\}, \{b\}]$, is eliminated by the constraint. However, it is not easy to understand this property, because in the "analogous" ASP program

$a \vee b.$

$\leftarrow not\, a.$

the constraint is indeed allowed, in ASP, to expunge from the (unique) world view $[\{a\}, \{b\}]$ of the bottom part (the set of its answer sets) the answer set $\{b\}$, thus producing for the program the unique world view $[\{a\}]$. This however, according to Subjective Constraint Monotonicity, should not be allowed for ELPs.

5. Conclusions

In this paper, we discussed Epistemic Logic Program (ELPs). We have presented a semantic approach for ELPs, called CF22, which applies in a straightforward way the underlying principles of the seminal ELP approach as presented and discussed by Gelfond in [2]. We devised CF22 not exactly to propose "yet another semantics", but rather in order to establish a principled way of comparing the different semantic approaches. We have augmented CF22 to CF22+S16C by adding a minimality criterion, S16C, "inherited" by the semantics S16 [7], that excludes some world views if there are others that rely on fewer assumptions about what is known.

We have experimented with CF22 on several examples taken from the relevant literature, for which the outcome of other semantic approaches was well-known. Results are quite surprising, as the new semantics does not agree uniformly with the others, and in some cases it agrees with none of them. More investigation is required to understand the reasons for these discrepancies. When CF22 agrees with S16 (which is often the case), it is not always needed to apply the S16C Criterion in order to get the same world views. As we may notice, a real novelty of our approach is that CF22 world views correspond to knowledge consistent sets of atoms, and this might presumably be a source of such differences.

CF22 can be taken as a basis for interesting extensions of the ELP paradigm. Precisely, in future work we intend to devise an extension where ELPs will be allowed to include rules with knowledge atoms as the head.

References

[1] M. Gelfond, H. Przymusinska, Definitions in epistemic specifications, in: A. Nerode, V. W. Marek, V. S. Subrahmanian (Eds.), Proc. of the 1st Intl. Workshop on Logic Programming and Non-monotonic Reasoning, The MIT Press, 1991, pp. 245–259.

[2] M. Gelfond, Logic programming and reasoning with incomplete information, Annals of Mathematics and Artificial Intelligence 12 (1994) 89–116. doi:10.1007/BF01530762.

[3] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. Kowalski, K. Bowen (Eds.), Proc. of the 5th Intl. Conf. and Symp. on Logic Programming, MIT Press, 1988, pp. 1070–1080.

[4] M. Gelfond, New semantics for epistemic specifications, in: J. P. Delgrande, W. Faber (Eds.), Proc. of LPNMR'11, volume 6645 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 260–265.

[5] M. Truszczynski, Revisiting epistemic specifications, in: M. Balduccini, T. C. Son (Eds.), Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, volume 6565 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 315–333.

[6] L. Fariñas del Cerro, A. Herzig, E. I. Su, Epistemic equilibrium logic, in: Q. Yang, M. J. Wooldridge (Eds.), Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, AAAI Press, 2015, pp. 2964–2970.

[7] Y. Shen, T. Eiter, Evaluating epistemic negation in answer set programming, Artificial Intelligence 237 (2016) 115–135.

[8] P. T. Kahl, A. P. Leclerc, Epistemic logic programs with world view constraints, in: A. Dal Palù, P. Tarau, N. Saeedloei, P. Fodor (Eds.), Tech. Comm. of ICLP 2018, volume 64 of *OASIcs*, Schloss Dagstuhl, 2018, pp. 1:1–1:17.

[9] E. I. Su, Epistemic answer set programming, in: F. Calimeri, N. Leone, M. Manna (Eds.), Proc. of JELIA'19, volume 11468 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 608–626.

[10] P. Cabalar, J. Fandinno, L. Fariñas del Cerro, Splitting epistemic logic programs, in: M. Balduccini, Y. Lierler, S. Woltran (Eds.), Proc. of LPNMR'19, volume 11481 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 120–133.

[11] P. Cabalar, J. Fandinno, L. Fariñas del Cerro, Splitting epistemic logic programs, Theory and Practice of Logic Programming 21 (2021) 296–316. doi:10.1017/S1471068420000058.

[12] P. Cabalar, J. Fandinno, L. Fariñas del Cerro, On the splitting property for epistemic logic programs (extended abstract), in: C. Bessiere (Ed.), Proceedings of IJCAI 2020, ijcai.org, 2020, pp. 4721–4725.

[13] P. Cabalar, J. Fandinno, L. Fariñas del Cerro, Autoepistemic answer set programming, Artificial Intelligence 289 (2020) 103382.

[14] V. Lifschitz, H. Turner, Splitting a logic program, in: Proc. of ICLP'94, MIT Press, 1994, pp. 23–37.

[15] V. Lifschitz, Thirteen definitions of a stable model, in: A. Blass, N. Dershowitz, W. Reisig (Eds.), Fields of Logic and Computation, volume 6300 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 488–503.

[16] S. Costantini, A. Formisano, Negation as a resource: a novel view on answer set semantics, Fundamenta Informaticae 140 (2015) 279–305.

[17] J. W. Lloyd, Foundations of Logic Programming, Springer-Verlag, 1987.

[18] S. Costantini, About epistemic negation and world views in epistemic logic programs, Theory and Practice of Logic Programming 19 (2019) 790–807. doi:10.1017/S147106841900019X.

[19] P. Kahl, R. Watson, E. Balai, M. Gelfond, Y. Zhang, The language of epistemic specifications (refined) including a prototype solver, J. Log. Comput. 30 (2015) 953–989. doi:10.1093/

logcom/exv065.

[20] E. I. Su, L. Fariñas del Cerro, A. Herzig, Autoepistemic equilibrium logic and epistemic specifications, Artificial Intelligence 282 (2020) 103249. doi:10.1016/j.artint.2020.103249.

[21] D. Pearce, A new logical characterization of stable models and answer sets, in: Non-Monotonic Extensions of Logic Programming, number 1216 in Lecture Notes in Artificial Intelligence, Springer, 1997, pp. 55–70.

[22] D. Pearce, A. Valverde, Synonymous theories in answer set programming and equilibrium logic, Proc. of ECAI04, 16th Europ. Conf. on Art. Intell. (2004) 388–390.

[23] V. Lifschitz, D. Pearce, A. Valverde, Strongly equivalent logic programs, ACM Transactions on Computational Logic 2 (2001) 526–541.

[24] D. Pearce, Equilibrium logic, Annals of Mathematics and Artificial Intelligence 47 (2006) 3–41. doi:10.1007/s10472-006-9028-z.

[25] E. I. Su, A monotonic view on reflexive autoepistemic reasoning, in: M. Balduccini, T. Janhunen (Eds.), Logic Programming and Nonmonotonic Reasoning - 14th Intl. Conf., LPNMR 2017, Proceedings, volume 10377 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 85–100.

[26] E. I. Su, Refining the semantics of epistemic specifications, in: A. F. et al. (Ed.), Proceedings 37th International Conference on Logic Programming, ICLP Technical Communications 2021, volume 345 of *EPTCS*, 2021, pp. 113–126. doi:10.4204/EPTCS.345.25.

Discovering Business Process models expressed as DNF or CNF formulae of Declare constraints

Federico Chesani[1,*,†], Chiara Di Francescomarino[2,†], Chiara Ghidini[2,†], Daniela Loreti[1,†], Fabrizio Maria Maggi[3,†], Paola Mello[1,†], Marco Montali[3,†], Elena Palmieri[1,†] and Sergio Tessaris[3,†]

[1]*DISI - University of Bologna, Italy*
[2]*Fondazione Bruno Kessler, Trento, Italy*
[3]*Free University of Bozen/Bolzano, Italy*

Abstract
In the field of Business Process Management, the Process Discovery task is one of the most important and researched topics. It aims to automatically learn process models starting from a given set of logged execution traces. The majority of the approaches employ procedural languages for describing the discovered models, but declarative languages have been proposed as well. In the latter category there is the Declare language, based on the notion of constraint, and equipped with a formal semantics on LTL*f*. Also, quite common in the field is to consider the log as a set of positive examples only, but some recent approaches pointed out that a binary classification task (with positive and negative examples) might provide better outcomes.

In this paper, we discuss our preliminary work on the adaptation of some existing algorithms for Inductive Logic Programming, to the specific setting of Process Discovery: in particular, we adopt the Declare language with its formal semantics, and the perspective of a binary classification task (i.e., with positive and negative examples).

Keywords
Process Discovery, Declare, Inductive Logic Programming

1. Introduction and motivations

The research field of Business Process Management (BPM) was initiated more than 20 years ago, and it is now a mature discipline that focuses on the many aspects related to the Business Processes and IT-solutions (but not only) for BPM. In particular, the mining of Business Processes (with the three main tasks of discovery, conformance checking and enhancement [1]) is a sub-field aimed to support decision-making in complex industrial and corporate domains. *Process*

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
*Corresponding author.
†These authors contributed equally.
federico.chesani@unibo.it (F. Chesani); dfmchiara@fbk.eu (C. Di Francescomarino); ghidini@fbk.eu (C. Ghidini); daniela.loreti@unibo.it (D. Loreti); maggi@inf.unibz.it (F. M. Maggi); paola.mello@unibo.it (P. Mello); montali@inf.unibz.it (M. Montali); e.palmieri@unibo.it (E. Palmieri); tessaris@inf.unibz.it (S. Tessaris)
0000-0003-1664-9632 (F. Chesani); 0000-0002-0264-9394 (C. Di Francescomarino); 0000-0003-1563-4965 (C. Ghidini); 0000-0002-6507-7565 (D. Loreti); 0000-0002-9089-6896 (F. M. Maggi); 0000-0002-5929-8193 (P. Mello); 0000-0002-8021-3430 (M. Montali); 0000-0001-5176-8843 (E. Palmieri); 0000-0002-3156-2669 (S. Tessaris)

discovery in particular deals with the automatic learning of a process model starting from a given set of logged traces, each one representing the execution of a business case. Accordingly to the language employed to represent the output process model, discovery algorithms fall into procedural or declarative techniques. The latter family of techniques—which represent the context of this work—return the model as a set of constraints (equipped with a declarative, logic-based semantics) that must be fulfilled by the traces at hand.

The Declare language [2] is one of the most used declarative languages, and consists of a set of template constraints that can be instantiated (grounded) with the process activities. The formal semantics of each constraint is based on LTL, and a process model is defined as a conjunction of grounded templates: hence, Declare does not (fully) support Conjunctive/Disjunctive Normal Forms. Moreover, the majority of the discovery algorithms conceive this task as a one-class supervised learning technique, while fewer works (e.g. [3, 4, 5, 6]) intend model-extraction as a two-class supervised task—provided that the log has been partitioned into two sets, usually named *positive* and *negative examples*.

In the field of Logic Programming, Inductive Logic Programming (ILP) is a well known family of learning techniques that address the learning task in terms of a binary classification problem. Noteworthy algorithms are the one proposed by Quinlan [7] and its subsequent generalization to DNF/CNF models proposed by Mooney [8]. There, the objective is to learn a logic-based description of two sets of ground facts.

In this paper, we discuss our preliminary investigations about the possibility of adapting the approach proposed by Mooney, to the specific setting of BP Discovery task, and Declare as the target language for describing the learned models. Hence, our approach will start from a log partitioned into two sets (positive and negative labeled traces), and the outcome will be a conjunction/disjunction of grounded Declare templates. The resulting model should be able then to discriminate positive from negative traces, as well as to properly classify novel traces. The proposed algorithms have been implemented in Prolog, and some preliminary testing has been done to evaluate the correctness of our approach, and the performances of the current implementation.

The paper is organized as follows: in Section 2 we provide some background on the field of Process Discovery and the Declare language, and on the original algorithms proposed by Mooney, from which we took inspiration. In Section 3 we introduce our extension/adaptation of the existing algorithms to the specific setting, and in Section 4 we experimentally evaluate our approach. In Section 5 we discuss some related works, while in Section 6 we discuss some conclusion remarks and future works.

2. Preliminaries

2.1. Process Discovery, and Declare

According to the Business Process Mining Manifesto [1], Process Discovery aims to "discover" a model of a process using the knowledge deduced from the event log without the use of a-priori information. A distinction between the many discovery algorithms can be done on the basis of the language adopted to output the learned process model. Indeed, two main categories of modeling languages can be easily identified: *procedural languages* and *declarative*

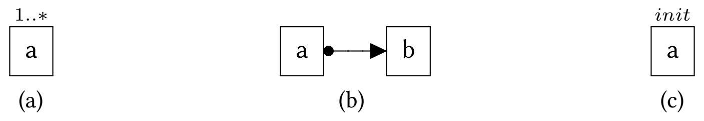


Figure 1: Examples of Declare constraints

languages. Procedural languages model the processes in terms of constructs like sequence, parallel executions, (exclusive) choices between different execution paths, etc. Declarative languages instead are more focused on the properties that each process execution should exhibit. While the former languages usually adopt a closed-approach (allowed execution paths are explicitly stated; everything else is forbidden by default), the latter approaches are usually based on a open-approach (whatever is not explicitly prohibited can be executed and is compliant with the process model). Notable examples of procedural modeling languages are YAWL [9] and BPMN [10, 11], while famous examples of declarative approaches are Declare [2] and Dynamic Condition Response Graphs [12].

Declare [2, 13] is one of the most well-established declarative process modeling languages. A process is modeled through a conjunction of constraints that affect the presence/absence of activity executions, and possibly the relative orders between activities. To this end, Declare provides a set of constraint templates that can be instantiated (grounded) by specifying the activities. Two main categories of constraint templates (or simply constraints, in the following) are available: *existence constraints* that involve only one activity, and *relation constraints*, that involve two of them. An example of an existence constraint is existence(a) (Figure 1a), that specifies that activity a must be executed at least once in every process instance. An example of relation constraint is response(a,b) (Fig. 1b): it states that, if activity a is executed, then it must be followed by the execution of activity b. Notice that the constraint is “triggered” by the execution of the activity a (graphically, a filled circle marks the triggering event), and that can be also vacuously satisfied if a is not executed. Each Declare template has been equipped with a formal semantics [2] in LTLf: for example, response(a,b) correspond to the expression $\Box(a \Rightarrow \Diamond b)$.

Some constraints are in a *subsumption* relation each other, meaning that traces satisfying a constraint will satisfy also another constraint, but not the opposite. For example, the init(a) constraint shown in Figure 1c states that every trace must begin with the execution of activity a: it is straightforward to see that every trace compliant with init(a) will be compliant also with existence(a), but not the other way round. Formally, as defined in [14], given a finite set A of activities, and A^* the set of sequences that can be generated from A, a constraint template C is *subsumed* by another constraint C’, written C $\sqsubseteq$ C’, if for every trace $t \in A^*$ and every parameter assignment γ_n from the parameters of C to tasks in A, whatever t complies with C/γ_n, then t also satisfies C’/γ_n. This hierarchy allows us to make specialization or generalization steps, as explained in the next section.

2.2. Learning CNF and DNF models: Mooney's approach

Mooney in [8], proposed two algorithms for learning Conjunctive and Disjunctive Normal Forms of logic models, respectively, starting from a labeled dataset. The DNF learner, called PFoil, is a propositional version of Quinlan's Foil [7], and it is composed of two cycles. The inner cycle focuses on the generation of terms, conjunctions of feature-value pairs that necessarily exclude all the negative examples in the event log. The outer cycle adds the returned clauses in disjunction to the model and ends when it covers all the positive traces. The next feature-value pair to add to the term is chosen calculating its *DNF gain*, a score based on the total number of positive and negative examples in the event log and on the number of covered ones. Intuitively, the best pair will be the one that covers more positive traces while excluding more negative ones.

Algorithm 1 PFoil: DNF learner by Mooney

```
Let Pos be all the positive examples.
Let DNF be empty.
Until Pos is empty do:
    Let Neg be all the negative examples.
    Set Term to empty and Pos2 to Pos.
    Until Neg is empty do:
        Choose the feature-value L that maximizes the function DNF-gain(L, Pos2, Neg).
        Add L to Term.
        Remove from Neg all the examples that do not satisfy L.
        Remove from Pos2 all the examples that do not satisfy L.
    Add Term as one term of DNF
    Remove from Pos all the examples that satisfy Term.
Return DNF.

Function DNF-gain(C, Pos, Neg)
    Let P be the number of examples in Pos
    Let N be the number of examples in Neg
    Let p be the number of examples in Pos that satisfy C
    Let n be the number of examples in Neg that satisfy C
    Return $p \times (\log_{10}(\frac{p}{p+n}) - \log_{10}(\frac{P}{P+N}))$.
```

The dual version of the algorithm outputs a CNF model. With respect to the Algorithm 1, the inner cycle focuses on the positives, while the outer cycle iterate over the negative exmaples. Consequently, also the gain function is adapted, and it is reported in Eq. 1.

$$\textit{CNF-gain} = n \times \left(log_{10} \frac{n}{p+n} - log_{10} \frac{N}{P+N} \right) \tag{1}$$

In this case though, p and n are the number of positive and negative traces that do not satisfy the feature-value pair in exam.

3. Applying Mooney's algorithm to the discovery of Declare process models

In this work we extend and adapt Mooney's algorithm to the process discovery task. With respect to the existing approaches for process discovery, here we consider the log as split in two (disjoint) classes of traces or, in other words, we conceive the discovery as a binary classification task. The goal, then, is to identify which are characteristics that allow us to discern if a not-labeled trace belongs to one class or another.

With respect to the original proposal by Mooney, we adapt the algorithm in several ways. First of all, the target language is Declare; secondly, the examples sets are indeed the traces belonging to a log, i.e. sequences of events that represent an execution of the process. Thirdly, when choosing the next constraint to be added in the resulting model, we introduce a further choice dimension (beside the gain function) by exploiting the subsumption relation between some Declare templates. Finally, we improve the algorithm for dealing with real logs and specific cases.

3.1. Declare as target language

Thanks to the declarative nature of Declare, and being the language based on the notion of constraints, it suffices to implement a specific test for checking when a trace satisfies or not a constraint. Our extended algorithm picks up constraints (rather than feature-value couples) from a list of candidates obtained by grounding the Declare constraint patterns.

Algorithm 2 Modified DNF Learner

```
Let Pos be all the positive traces.
Let DNF, ExcludedNeg and ExcludedPos be empty.
Until Pos is empty do:
    Let Neg be all the negative traces.
    Set Term to empty and Pos2 to Pos.
    Until Neg is empty do:
        If the list of candidate constraints is not empty:
            Choose the constraint C that maximizes the function DNF-gain(C, Pos2, Neg).
            Add C to Term.
            Remove from Neg all the traces that do not satisfy C.
            Remove from Pos2 all the traces that do not satisfy C.
        else:
            Set ExcludedNeg to Neg and Neg to empty.
    Remove from Pos all the traces that satisfy Term.
    If at least one positive trace satisfies Term:
        Add Term as one term of DNF
    Else:
        Set ExcludedPos to Pos and Pos to empty.
Return DNF.
```

The constraint that maximizes the gain function is chosen using Mooney's formula. In the DNF version of the algorithm, then, the chosen constraint is added to Term (in conjunction), thus performing a "specialization" step, since the resulting conjunction of constraints will exclude more negative traces but (possibly) also more positive ones. Similarly, the CNF algorithm will perform a generalization step, since adding a constraint in a disjunction will allow to possibly accept more positive and/or negative traces.

In the DNF algorithm, the inner cycle outputs a term that is a conjunction of constraints that rules out all the negative examples. It is possible, however, that a term rules out also all the positive examples. In such a case, the term would be redundant in the final model. We add a control step that checks if at least one positive example is satisfied, and only in that case the term is added to the model. Moreover, if a term does not allow any positive trace, the algorithm will never be able to find another term (if there exists such a term, the gain function would have selected proper constraints in earlier iterations). Therefore, it is wiser to stop the computation and to ignore the remaining positive traces. The same thing can happen in the CNF algorithm; if a clause, that has to satisfy all the positive traces, does not exclude any negative one, then it is discarded.

Example 1 shows a simple log, and one among the many possible Declare models. Each trace is represented as a Prolog structure, where the first argument is the trace identifier, while the second is a list of events. In turn, each event is described by the name of the process activity that has been executed, and the timestamp (an integer).

Example 1. *Traces labeled as positive examples:*
trace(tp1, [event(a,1), event(b,2), event(c,3), event(d,4)]).
trace(tp2, [event(a,1), event(b,2), event(b,3), event(c,4)]).
trace(tp3, [event(a,1), event(c,2), event(b,3), event(d,4)]).
trace(tp4, [event(k,1), event(c,2), event(a,3), event(d,4)]).
Traces labeled as negative examples:
trace(tn1, [event(b,1), event(c,2), event(e,3), event(d,4)]).
trace(tn2, [event(c,1), event(b,2), event(a,3), event(a,4)]).
A possible DNF model could be:

(existence(a) AND precedence(a,b)) OR existence(k)

Analogously, a CNF model would be:

(existence(a) OR existence(k)) AND precedence(a,b)

□

3.2. Exploiting the subsumtption hierarchy

Some Declare templates are in a subsumption relation with each other, as pointed out in [14]. For example, the init(a) constraint imposes that each trace should begin with the execution of activity a; consequently, any trace that satisfies such constraint will satisfies also the more

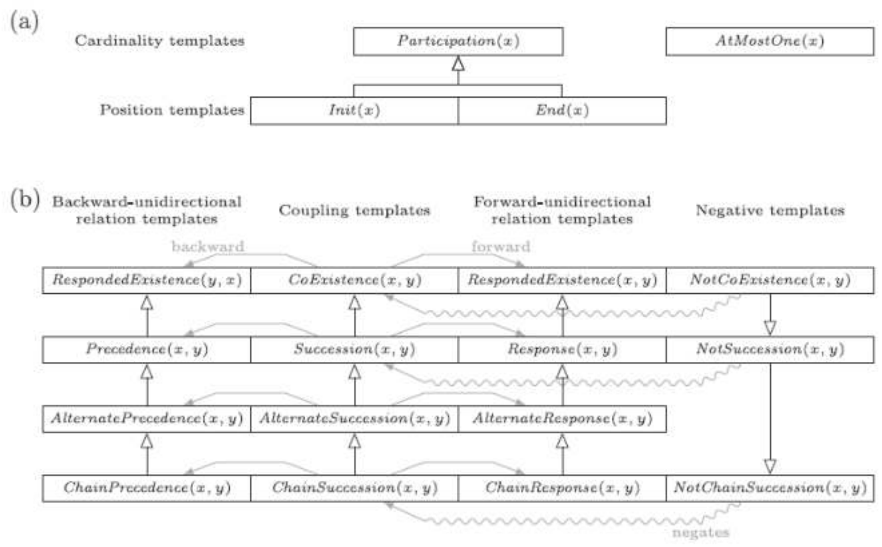


Figure 2: Subsumption map of Declare templates [14]. Note that *Participation(x)* corresponds to the template *existence(X)*, *End(x)* corresponds to *last(X)*, and *AtMostOne(x)* to *absence2(X)*

.

general constraint existence(a). In Figure 2 we report the subsumption hierarchy proposed in [14]. We exploit such relations in two ways.

First of all, the gain function will select candidates starting from two initial sets of constraints. As the inner cycle of the DNF specializes the current term, its starting set will contain the more general templates, accordingly to the sumbsumption hierarchy. Analogously, the CNF version will start considering the more specialized constraints.

Secondly, the specialization step (in the DNF algorithm) is extended as well: beside adding a new constraint in conjunction to the term, the subsumption relation allows us to specialize a constraint already in the term. Suppose for example that the current term constructed by the inner cycle is (existence(a)). The specialization step could then opt to add a new constraint in conjunction, for example responded_existence(b,c), or specialize the existing one, for instance, in init(a). The resulting models would be [existence(a) AND responded_existence(b,c)] in the former case, and [init(a)] in the latter.

Analogous consideration hold for the CNF algorithm, with the obvious difference that the subsumption hierarchy is explored towards the generalization.

3.3. Dealing with real-life logs

It might not be always possible to "perfectly" separate positive form negative examples. This because of two possible reasons: the Declare language provides a bias about the allowed LTL formulas, and to the best of our knowledge, there is not any proof of completeness of such language w.r.t. the classification task. Moreover, real-life logs might be inconsistent, i.e. a trace might have been labeled as positive and as negative at the same time. To cope with these exceptions, whenever our algorithms find negative traces that are impossible to exclude or positive ones that cannot be covered, they simply remove them from the event log under consideration and continue with the discovery task. At the end, the ignored traces are returned together with the found model, if any.

Sometimes real-life logs contains positive traces only, and no negative examples are provided. Such case affects both the DNF and the CNF original algorithms, as the DNF version's termination condition is given by the set of negative examples becoming empty, whereas the CNF version would be stuck in an infinite loop as it would generate empty clauses. We designed our algorithm to be able to deal with such logs, and to return process models that just describe the positive traces.

4. Experimental evaluation

We implemented both the DNF and the CNF algorithms in Prolog. The core of both the algorithms is the predicate that chooses the next constraint to be added to the term. In the DNF version, at the first iteration over a term, the predicate chooses from the set of the most general constraints. In the subsequent iterations it will choose between the most general constraints (not yet in the term) and the specialization of an already selected constraint. In the CNF version, the same will happen, a part that the specialization step will bu substituted by the generalization one.

We evaluated both the algorithms against two logs: a synthetic, controlled log whose process model was already known, and a real-life event log (about a PAP test screening process), whose model was not known in advance.

4.1. The synthetic, controlled event log

The controlled event log contains a set of 64000 positive traces and three different sets containing respectively 10240, 12800 and 25600 negative ones, with 16 different activities. Each one of the negative example sets violates a single, specific constraint: hence for each log a constraint would be enough to discriminate between the positive and the negative traces.

All the negative examples contained in the first negative set can be ruled out by the constraint exclusive_choice(send_acceptance_pack, receive_negative_feedback). Both the DNF and the CNF algorithms returned the same model:

```
exclusive_choice(send_acceptance_pack, receive_negative_feedback)
```

The two remaining negative sets violate a precedence constraint grounded over different activities, affecting the overall process in different manner. The precedence(X, Y) template however is neither in the starting set of the DNF algorithm nor in the one of the CNF version. The second set of negative traces was found to be completely ruled out by not_chain_succession(assess_loan_risk, appraise_property) and chain_succession(receive_loan_application, appraise_property), which were the models returned by the algorithm, and indeed are correct w.r.t. the original violated constraint. The model returned by the CNF version with the third set of negative traces contained the expected constraint. On the other hand, the DNF version returned a correct but more complex model, composed of three constraints.

	DNF version	CNF version
Negative Set #1	1129	297
Negative Set #2	700	307
Negative Set #3	1648	522

Table 1
Average time (in seconds) of execution for the controlled event log

From a performance point of view, as visible in Table 1, the CNF version of the algorithm was always faster than the DNF one. This might be the consequence of the fact that, roughly speaking, the CNF and DNF algorithms proceed with the specialization/generalization of the returned model: thus, they start to explore different initial constraints. Figures 3 and 4 show the occupation of the global and local stacks right before the termination of the algorithms, when the final model has already been found.

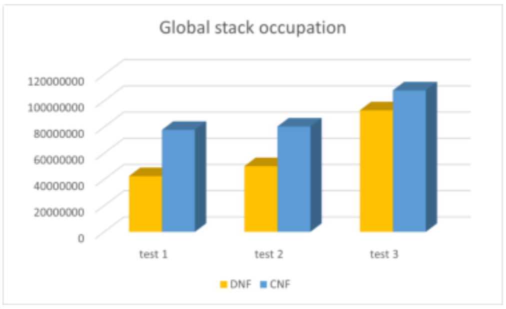


Figure 3: Global stack occupation right before the termination of the algorithms.

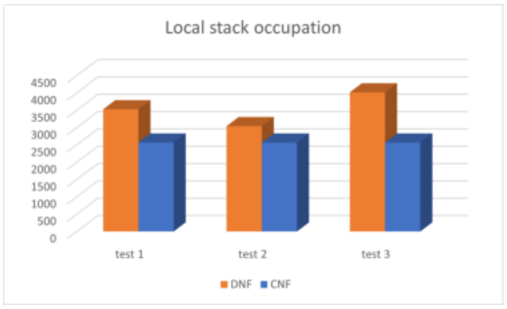


Figure 4: Local stack occupation right before the termination of the algorithms.

4.2. A real-life event log: the PAP test

Once performances have been assessed through the use of a synthetic controlled log, we evaluated the algorithms on a real-life event log that contains traces relative to PAP test screenings. The number of activities in this log is slightly higher than in the controlled event log (19 vs. 16), but the number of traces is way lower as there are only 55 positive examples and 102 negative ones. The discovered DNF and CNF models were respectively:

```
choice(refuse, send_result_inadequate_papTest)
OR
(
  exactly1(send_letter_negative_papTest)
  AND
  choice(send_letter_negative_papTest, execute_colposcopy_exam)
)
```

And:

```
(
  exclusive_choice(send_letter_negative_biopsy,
      send_result_inadequate_papTest)
  OR
  chain_succession(phone_call_positive_papTest,
      execute_colposcopy_exam)
)
AND
(
  chain_succession(invite, refuse)
  OR
  chain_succession(invite, execute_papTest_exam)
)
```

Regarding the performances, the discovered models were returned in a very short time as the number of traces is extremely lower than the one in the controlled event log. Again, the performance of the CNF version is better than the DNF's one, with respectively 3 and 5 seconds taken on average to return the model.

5. Related works

Traditional process discovery approaches aim at extracting a process model from positive examples of business executions. As pointed out by Goedertier et al. [4], they can be interpreted as machine learning techniques to extract a grammar from a set of positive sample data [15]. However, in process discovery authors typically make use of formalisms to express concurrency and synchronization in a more understandable way w.r.t. grammar learning (where automata, regular expressions or production rules are often employed to represent the model). Since the learning task is inevitably influenced by the type of language used for the model, this element is often used to classify process discovery approaches into two macro-categories: procedural and declarative.

Procedural approaches envisage to uncover structured processes [16, 17, 18, 19, 20, 21, 22, 23]. For the sake of our comparison, it is also important to underline that most of these works contemplate the presence of negative information in the log in the shape of non-informative noise, which should be discarded. The approach in this work is instead an example to declarative process discovery Since process models are sometimes less structured than one could expect [24], procedural discovery can lead to the identification of spaghetti-models. Declarative approaches [25, 26, 24, 27, 28, 14, 29] aim at overcoming this issue by offering a compact way to briefly list the required or prohibited behaviours in a business process. Similarly to the procedural approaches, the declarative discoverers listed so far do not deal with negative examples. Nonetheless, they indirectly envisage the possibility to discard a portion of the log by setting thresholds on metrics that the discovered model should satisfy.

In the field of grammar learning, Gold [30] showed how both positive and negative examples are required to discover a grammar with perfect accuracy. A claim particularly relevant to our work is that, in order to distinguish the right hypothesis among an infinite number of grammars that fit the positive examples, the key element is the availability of negative examples. The reason why many procedural and declarative discoverers do not take into account negative examples can be identified in the fact that these approaches do not usually seek perfection, but focus on good performance according to defined metrics. Among traditional grammar learning approaches, the ones by Angluin [31] and Mooney [8] are particularly relevant for our work. The article [31] focuses on identifying an unknown model referred as "regular set" and represented through Deterministic Finite-state Acceptor (DFA). Coherently with Gold's theory [30], Angluin propose a learning algorithm that starts from input examples of the regular set's members and non-members. The learning process is realised through the construction of an "observation table". As discussed in this article, the approach of Mooney et al. [8] shows instead three different algorithms to learn CNF, DNF and decision trees from a set of positive and negative examples.

A subset of the declarative discoverers [32, 33, 5, 34, 35] is related to the basic principles of Inductive Constraint Logic (ICL) [36]–which depends on the availability of both negative and positive examples. In particular, similarly to the approach of this work, DecMiner [5] starts from an input set of labelled examples and learns a set of SCIFF rules [37], subsequently translated into ConDec constraints [38]. Differently from [5], our approach avoids intermediate language and aims at learning Declare constraints directly. The work [6] by Slaats et al. propose a universal declarative miner, applicable but not limited to Declare language, which makes use of negative and positive traces. W.r.t. our work, the generality of the approach in [6] hiders the use of subsumption to avoid redundancy and identify the most general model. Other relevant works are those of Neider et al. [39], Camacho et al. [40], and Reiner [41], which start form an input data set of positive and negative examples and employ a SAT-based solver to learn a simple set of LTL formulas. Differently from these works, we opt for Declare formulas with LTL$_f$ semantics. In [42], a SAT-based solver is also used to discover a Declare model from a log with both positive and negative traces. According to the classification of Gunther et al. [43], the concept of negative example used in these works (as well as in this one) is connected to both the concepts of syntactical and semantic noise.

Another research field related to our work is that of deviance mining [44], which aims at characterizing those log traces that deviates from the expected behaviour. In particular, some works focus on the differences between the models discovered from deviant and non-deviant traces [45, 46], whereas other works intend deviance mining as a classification task similarly to sequence classification [47, 48, 49, 50, 51, 52, 53].

A limited number of recently proposed procedural approaches [54, 4, 55, 56, 57] also actively take into account negative examples. Finally, the development of synthetical log generators producing both positive and negative process cases [58, 59, 60, 4, 61, 62] is another sign that underlines how the process discovery research field is increasingly considering negative traces as informative examples.

6. Conclusions and future work

In this work we presented an adaptation of well known discovery algorithms from previous works [7, 8]. In particular, we chose as target language for process descriptions the Declare language: being based on a formal semantics expressed in LTL*f*, the adaptation of the existing approaches was quite seamless. Then, we exploited the existence of a subsumption relation between some Declare templates to extend the specialization/generalization steps towards in the original algorithms.

From the perspective of the BPM research field, and of the Declare-based approaches, it is worthy to notice that our discovery algorithms are quite innovative w.r.t. existing approaches. Firstly because we put a strong emphasis on the use of negative examples. Secondly, and more important, because we suggest new Declare models based on DNF or CNF formulas. Indeed, at its core, Declare allows only models defined in terms of conjunction of constraints, and the disjunction is not fully supported. Hence, Declare in its original definition would not support CNF models, nor DNF, as instead we do in this paper. It is highly debatable, however, if the introduction of full DNF/CNF models allows to obtain simpler, or more meaning process models w.r.t. the original limitations imposed by Declare. In turn, usability of the whole system might be affected by the type of discovered model. These are indeed topics of future investigation.

Besides this, there are many aspects that we plan to investigate in future research activities. First of all, Declare models are defined in terms of completely grounded constraints: the introduction of variables in the constraints might result in better process models, and technically speaking, Inductive Logic Programming algorithms would provide already an interesting solution (at a higher computational cost, unfortunately). The use of variables would also offer another way for specializing/generalizing the models.

Another aspect that might enjoy the use of variables in the models, and the adoption of ILP techniques, is related to the presence of data in the logs. It is quite common to encounter process logs where activities in a trace are associated with more information than just their name or timestamp. Being able to support all the data associated to each activities could make it possible to perform other tasks, different from the generation of the model. For example, having not only the information that someone logged into their account at a certain time, but also knowing who it was and what password was used could be useful for a statistical research on how many times someone tried to log into a certain profile, leading to the identification of hacking attempts and of the processes adopted in the attempts.

From a technical viewpoint, the introduction of variables in the Declare model would require a different semantics (the current one is based on propositional LTL over finite traces). In this sense, Constraint Logic Programming (CLP) over finite domains might be a viable alternative, supporting the semantics and the implementation at the same time. With a minimum amount of code it would be feasible to specify, for example, that a certain variable "X" can only assume values associated with a finite set of activities. As a side advantage, the definition of some constraints would be easier: for example, a quite common business constraint is that a certain activity should not be executed twice consecutively; this would be achieved through a chain_response(X,Y) constraint, with a further CLP constraint $\mathsf{X} \neq \mathsf{Y}$.

Another interesting research direction regards the gain function used in the algorithm. Currently, the gain function only takes into account the number of positive and negative traces covered by a constraint. However, we might imagine scenarios where the users want to express desiderata and preferences over the discovered process models. For example, users might have a preference for a specific constraint template like init(X), or constraints grounded on certain activities rather than others. This could be achieved by defining a different gain function or, exploiting the existing literature on the topic, by investigating the relations with the existing preference logics. In turn, such perspective open up a question about the optimality of a model, that indeed was not investigated in this work.

Finally, a deeper comparison with existing approaches should be carried on, in order to better understand the quality of the discovered models, the usability of the approach and the usability of the discovered models from the final user viewpoint, and also to assess the performances of our approach w.r.t. state-of-the-art process discovery algorithms.

Acknowledgments

This work has been partially supported by the European Union's H2020 projects HumaneAI-Net (g.a. 952026), StairwAI (g.a. 101017142), and TAILOR (g.a. 952215).

References

[1] W. M. P. van der Aalst, al., Process mining manifesto, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), BPM Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, 2011, Revised Selected Papers, Part I, volume 99 of *LNBIP*, Springer, 2011, pp. 169–194. doi:10.1007/978-3-642-28108-2_19.

[2] M. Pesic, Constraint-based workflow management systems : shifting control to users, Ph.D. thesis, Industrial Engineering and Innovation Sciences, 2008. doi:10.6100/IR638413.

[3] L. Maruster, A. J. M. M. Weijters, W. M. P. van der Aalst, A. van den Bosch, A rule-based approach for process discovery: Dealing with noise and imbalance in process logs, Data Min. Knowl. Discov. 13 (2006) 67–87.

[4] S. Goedertier, D. Martens, J. Vanthienen, B. Baesens, Robust process discovery with artificial negative events, J. Mach. Learn. Res. 10 (2009) 1305–1340.

[5] F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, S. Storari, Exploiting inductive logic programming techniques for declarative process mining, Trans. Petri Nets Other Model. Concurr. 2 (2009) 278–295.

[6] T. Slaats, S. Debois, C. O. Back, Weighing the pros and cons: Process discovery with negative examples, in: BPM, volume 12875 of *LNCS*, Springer, 2021, pp. 47–64.

[7] J. R. Quinlan, Learning logical definitions from relations, Mach. Learn. 5 (1990) 239–266. URL: https://doi.org/10.1007/BF00117105. doi:10.1007/BF00117105.

[8] R. J. Mooney, Encouraging experimental results on learning CNF, Mach. Learn. 19 (1995) 79–92.

[9] M. Adams, A. V. Hense, A. H. ter Hofstede, Yawl: An open source business process management system from science for science, SoftwareX 12 (2020) 100576. doi:https://doi.org/10.1016/j.softx.2020.100576.

[10] Business process model and notation (BPMN), https://www.bpmn.org/, 2022.

[11] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, On the suitability of BPMN for business process modelling, in: S. Dustdar, J. L. Fiadeiro, A. P. Sheth (Eds.), BPM, 4th Intl. Conf., BPM 2006, Vienna, Austria, September 5-7, 2006, Procs., volume 4102 of *LNCS*, Springer, 2006, pp. 161–176. doi:10.1007/11841760_12.

[12] T. T. Hildebrandt, R. R. Mukkamala, Declarative event-based workflow as distributed dynamic condition response graphs, in: K. Honda, A. Mycroft (Eds.), Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES 2010, Paphos, Cyprus, 21st March 2010, volume 69 of *EPTCS*, 2010, pp. 59–73. URL: https://doi.org/10.4204/EPTCS.69.5. doi:10.4204/EPTCS.69.5.

[13] W. M. P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Comput. Sci. Res. Dev. 23 (2009) 99–113.

[14] C. D. Ciccio, F. M. Maggi, M. Montali, J. Mendling, Resolving inconsistencies and redundancies in declarative process models, Inf. Syst. 64 (2017) 425–446.

[15] D. Angluin, C. H. Smith, Inductive inference: Theory and methods, ACM Comput. Surv. 15 (1983) 237–269.

[16] A. J. M. M. Weijters, W. M. P. van der Aalst, Rediscovering workflow models from event-based data using little thumb, Integr. Comput. Aided Eng. 10 (2003) 151–162.

[17] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (2004) 1128–1142.

[18] C. W. Günther, W. M. P. van der Aalst, Fuzzy mining - adaptive process simplification based on multi-perspective metrics, in: BPM, volume 4714 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 328–343.

[19] W. M. P. van der Aalst, V. A. Rubin, H. M. W. Verbeek, B. F. van Dongen, E. Kindler, C. W. Günther, Process mining: a two-step approach to balance between underfitting and overfitting, Software and Systems Modeling 9 (2010) 87–111.

[20] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering block-structured process models from event logs - A constructive approach, in: Petri Nets, volume 7927 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 311–329.

[21] Q. Guo, L. Wen, J. Wang, Z. Yan, P. S. Yu, Mining invisible tasks in non-free-choice constructs, in: BPM, volume 9253 of *LNCS*, Springer, 2015, pp. 109–125.

[22] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, Split miner: Discovering accurate and simple business process models from event logs, in: ICDM, IEEE Computer Society, 2017, pp. 1–10.

[23] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, A. Soo, Automated discovery of process models from event logs: Review and benchmark, IEEE Trans. Knowl. Data Eng. 31 (2019) 686–705.

[24] F. M. Maggi, R. P. J. C. Bose, W. M. P. van der Aalst, Efficient discovery of understandable declarative process models from event logs, in: CAiSE, volume 7328 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 270–285.

[25] F. M. Maggi, A. J. Mooij, W. M. P. van der Aalst, User-guided discovery of declarative process models, in: CIDM, IEEE, 2011, pp. 192–199.

[26] W. M. P. van der Aalst, H. T. de Beer, B. F. van Dongen, Process mining and verification of properties: An approach based on temporal logic, in: OTM Conferences (1), volume 3760 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 130–147.

[27] D. M. M. Schunselaar, F. M. Maggi, N. Sidorova, Patterns for a log-based strengthening of declarative compliance models, in: IFM, volume 7321 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 327–342.

[28] C. D. Ciccio, M. H. M. Schouten, M. de Leoni, J. Mendling, Declarative process discovery with minerful in prom, in: BPM (Demos), volume 1418 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 60–64.

[29] C. O. Back, T. Slaats, T. T. Hildebrandt, M. Marquard, DisCoveR: accurate and efficient discovery of declarative process models, Int. J. Softw. Tools Technol. Transfer (2021).

[30] E. M. Gold, Language identification in the limit, Inf. Control. 10 (1967) 447–474.

[31] D. Angluin, Learning regular sets from queries and counterexamples, Inf. Comput. 75 (1987) 87–106.

[32] E. Lamma, P. Mello, F. Riguzzi, S. Storari, Applying inductive logic programming to process mining, in: ILP, volume 4894 of *LNCS*, Springer, 2007, pp. 132–146.

[33] E. Lamma, P. Mello, M. Montali, F. Riguzzi, S. Storari, Inducing declarative logic-based models from labeled traces, in: G. Alonso, P. Dadam, M. Rosemann (Eds.), Business Process Management, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 344–359.

[34] E. Bellodi, F. Riguzzi, E. Lamma, Probabilistic logic-based process mining, in: CILC, volume 598 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2010.

[35] E. Bellodi, F. Riguzzi, E. Lamma, Statistical relational learning for workflow mining, Intell. Data Anal. 20 (2016) 515–541.

[36] L. D. Raedt, W. V. Laer, Inductive constraint logic, in: ALT, volume 997 of *Lecture Notes in Computer Science*, Springer, 1995, pp. 80–94.

[37] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, Verifiable agent interaction in abductive logic programming: The SCIFF framework, ACM Trans. Comput. Log. 9 (2008) 29:1–29:43.

[38] M. Pesic, W. M. P. van der Aalst, A declarative approach for flexible business processes management, in: Business Process Management Workshops, volume 4103 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 169–180.

[39] D. Neider, I. Gavran, Learning linear temporal properties, in: FMCAD, IEEE, 2018, pp. 1–10.

[40] A. Camacho, S. A. McIlraith, Learning interpretable models expressed in linear temporal logic, in: ICAPS, AAAI Press, 2019, pp. 621–630.

[41] H. Riener, Exact synthesis of LTL properties from traces, in: FDL, IEEE, 2019, pp. 1–6.

[42] F. Chesani, C. D. Francescomarino, C. Ghidini, D. Loreti, F. M. Maggi, P. Mello, M. Montali, S. Tessaris, Process discovery on deviant traces and other stranger things, IEEE Transactions on Knowledge and Data Engineering (2021). Under review.

[43] C. W. Günther, Process Mining in Flexible Environments, Ph.D. thesis, Technische Universiteit Eindhoven, 2009.

[44] H. Nguyen, M. Dumas, M. L. Rosa, F. M. Maggi, S. Suriadi, Business process deviance mining: Review and evaluation, CoRR abs/1608.08252 (2016).

[45] S. Suriadi, R. Mans, M. T. Wynn, A. Partington, J. Karnon, Measuring patient flow variations: A cross-organisational process mining approach, in: AP-BPM, volume 181 of *Lecture Notes in Business Information Processing*, Springer, 2014, pp. 43–58.

[46] A. Armas-Cervantes, P. Baldan, M. Dumas, L. García-Bañuelos, Behavioral comparison of

process models based on canonically reduced event structures, in: BPM, volume 8659 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 267–282.

[47] S. Suriadi, M. T. Wynn, C. Ouyang, A. H. M. ter Hofstede, N. J. van Dijk, Understanding process behaviours in a large insurance company in australia: A case study, in: CAiSE, volume 7908 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 449–464.

[48] A. Partington, M. T. Wynn, S. Suriadi, C. Ouyang, J. Karnon, Process mining for clinical processes: A comparative analysis of four australian hospitals, ACM Trans. Management Inf. Syst. 5 (2015) 19:1–19:18.

[49] J. Swinnen, B. Depaire, M. J. Jans, K. Vanhoof, A process deviation analysis - A case study, in: Business Process Management Workshops (1), volume 99 of *Lecture Notes in Business Information Processing*, Springer, 2011, pp. 87–98.

[50] R. P. J. C. Bose, W. M. P. van der Aalst, Discovering signature patterns from event logs, in: CIDM, IEEE, 2013, pp. 111–118.

[51] D. Lo, S. Khoo, C. Liu, Efficient mining of iterative patterns for software specification discovery, in: KDD, ACM, 2007, pp. 460–469.

[52] M. L. Bernardi, M. Cimitile, C. Di Francescomarino, F. M. Maggi, Do activity lifecycles affect the validity of a business rule in a business process?, Inf. Syst. 62 (2016) 42–59.

[53] J. D. Smedt, G. Deeva, J. D. Weerdt, Mining behavioral sequence constraints for classification, IEEE Trans. Knowl. Data Eng. 32 (2020) 1130–1142.

[54] H. P. de León, C. Rodríguez, J. Carmona, K. Heljanko, S. Haar, Unfolding-based process discovery, in: ATVA, volume 9364 of *LNCS*, Springer, 2015, pp. 31–47.

[55] H. P. de León, L. Nardelli, J. Carmona, S. K. L. M. vanden Broucke, Incorporating negative information to process discovery of complex systems, Inf. Sci. 422 (2018) 480–496.

[56] H. M. Ferreira, D. R. Ferreira, An integrated life cycle for workflow management based on learning and planning, Int. J. Cooperative Inf. Syst. 15 (2006) 485–505.

[57] S. K. L. M. vanden Broucke, J. D. Weerdt, J. Vanthienen, B. Baesens, Determining process model precision and generalization with weighted artificial negative events, IEEE Trans. Knowl. Data Eng. 26 (2014) 1877–1889.

[58] F. Chesani, C. Di Francescomarino, C. Ghidini, D. Loreti, F. M. Maggi, P. Mello, M. Montali, V. Skydanienko, S. Tessaris, Towards the generation of the "perfect" log using abductive logic programming, in: CILC, volume 2396 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 179–192.

[59] F. Chesani, A. Ciampolini, D. Loreti, P. Mello, Abduction for generating synthetic traces, in: BPM Workshops, volume 308 of *LNBIP*, Springer, 2017, pp. 151–159.

[60] D. Loreti, F. Chesani, A. Ciampolini, P. Mello, Generating synthetic positive and negative business process traces through abduction, Knowl. Inf. Syst. 62 (2020) 813–839.

[61] T. Stocker, R. Accorsi, Secsy: A security-oriented tool for synthesizing process event logs, in: BPM (Demos), volume 1295 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014, p. 71.

[62] K. M. van Hee, Z. Liu, Generating benchmarks by random stepwise refinement of petri nets, in: ACSD/Petri Nets Workshops, volume 827 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2010, pp. 403–417.

Efficient Theorem Proving for Conditional Logics with Conditional Excluded Middle

Nikola Panic[1], Gian Luca Pozzato[1]

[1]Dipartimento di Informatica, Università di Torino, Italy

Abstract

In this work we introduce a labelled sequent calculus for Conditional Logics admitting the axiom of Conditional Excluded Middle (CEM), rejected by Lewis but endorsed by Stalnaker. We also consider some of its standard extensions. Conditional Logics with CEM recently have received a renewed attention and have found several applications in knowledge representation and artificial intelligence. The proposed calculus improves the only existing one, SeqS, where the condition CEM on conditional models is tackled by means of a simple but computationally expensive process of label substitution. Here we propose an alternative calculus avoiding label substitution, where a single rule deals simultaneously with conditional formulas and the CEM axiom. We have implemented the calculi in Prolog following the "lean" methodology, then we have tested the performances of the prover and compared them with those of CondLean, an implementation of SeqS. The performances are promising and better than those of CondLean, witnessing that the proposed calculus provides an effective improvement with respect to the state of the art.

Keywords

conditional logics, sequent calculi, proof methods, theorem proving

1. Introduction

Conditional logics are extensions of classical logic by means of a binary operator $>$, in order to express conditional implications of the form $A > B$. They have a long history, starting with the seminal works by [1], [2], [3], [4], and [5]. Conditional logics have found an interest in several fields of artificial intelligence and knowledge representation, from reasoning about prototypical properties and non-monotonic reasoning [6, 7, 8, 9], where $A > B$ can be used to formalize that "typically, the As are also Bs" or "in normal circumstances, if A then B", to modeling belief change, knowledge update and revision [10, 11, 12], where the relation with conditional logics is expressed by the so-called *Ramsey's Rule*:

$$(A \circ B) \rightarrow C \text{ holds} \quad \textit{if and only if} \quad A \rightarrow (B > C) \text{ holds}$$

where the operator $\circ$ is any *update* operator satisfying postulates of [13], that are considered the "core" properties for any concrete and plausible operator of belief update. Ramsey's rule means that C is entailed by "A updated by B" if and only if the conditional $B > C$ is entailed by A. In this sense it can be said that the conditional $B > C$ expresses an hypothetical update

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
nikola.panic@edu.unito.it (N. Panic); gianluca.pozzato@unito.it (G. L. Pozzato)
0000-0002-3952-4624 (G. L. Pozzato)

of the information A. Moreover, conditional logics have been employed in order to represent conditional sentences that cannot be captured by material implication and, in particular, *counterfactuals* [1], e.g., conditionals of the form "if A were the case, then B would be the case", where A is false, as well as to model hypothetical queries in deductive databases and logic programming [14], causal inference and reasoning about action execution in planning [15, 16], access control policies in security [17].

Similarly to modal logics, the semantics of conditional logics can be defined in terms of possible world structures. In this respect, conditional logics can be seen as a generalization of modal logics (or a type of multi-modal logic) where the conditional operator is a sort of modality indexed by a formula of the same language. However, as a difference with modal logics, a universally accepted semantics for conditional logics lacks and it is the main reason for the underdevelopment of proof-methods and theorem provers. The semantics we consider in this work is the *selection function semantics* [2], where truth values are assigned to formulas depending on a world. Intuitively, the selection function f selects, for a world w and a formula A, the set of worlds $f(w, A)$ which are "most-similar to w" or "closer to w" given the information A. In *normal* conditional logics, the function f depends on the set of worlds satisfying A rather than on A itself, so that $f(w, A) = f(w, A')$ whenever A and A' are true in the same worlds (normality condition). A conditional sentence $A > B$ is true in w whenever B is true in every world selected by f for A and w. It is the normality condition which marks essentially the difference between conditional logics on the one hand, and multimodal logic, on the other (where one might well have a family of $\Box$ indexed by formulas). We believe that it is the very condition of normality what makes it difficult to develop proof systems for conditional logics with the selection function semantics.

Since we adopt the selection function semantics, CK is the fundamental system [2]; it has the same role as the system K (from which it derives its name) in modal logic: CK-valid formulas are exactly those ones that are valid in every selection function model. Extensions are then obtained by imposing restrictions on the selection function. In this work, we focus on the systems equipped with the condition of *Conditional Excluded Middle* (CEM), whose characterizing axioms are of the form

$$(A > B) \vee (A > \neg B)$$

corresponding to the semantic condition that, for each world w and for each formula A, the selection function f selects at most one world for w and A, in other words the cardinality of the selection function is at most 1.

While [1] provides an argument against CEM, essentially based on his treatment of "might" counterfactuals so that both conditionals can be false, [3] provides an argument in favor, intuitively stating that the conditionals can be *indeterminate* but their disjunction is true. Consider the example in [18] and the two counterfactual sentences "if Bizet and Verdi were compatriots, would they be Italian?" and "if Bizet and Verdi were compatriots, would they be not Italian?": Lewis rejects, stating that the two conditionals are intuitively false, whereas Stalnaker endorses it, conjecturing that they are both indeterminate but their disjunction is true. More recently, [18] has provided a general positive argument for CEM, defending the Stalnaker's verdict.

In [19] the authors have introduced a labelled sequent calculus for CK and the extensions with condition CEM, but also ID (identity), MP (conditional modus ponens), and CS (conditional strong centering), as well as most of the combinations of them. The proposed calculi, called SeqS, are modular and, in some cases, optimal, however, for the systems with CEM, a label substitution mechanism is needed in order to deal with the above mentioned condition on the selection function. They have also introduced a Prolog theorem prover, called CondLean, implementing those calculi, whose performances are promising in general, however, due to the label substitution mechanism, they degrade for systems with CEM, especially in finding that a formula is *not* valid.

In this paper we provide a first step in the direction of efficient theorem proving for conditional logics dealing with conditional excluded middle, by tackling the problems of SeqS and CondLean with an alternative calculus (and, as a consequence, an alternative implementation) in which the label substitution mechanism is replaced by a suitable rule for dealing with conditional formulas in these systems. We are able to give cut-free calculi, called SeqS', for CK+CEM and all the extensions with ID and CS. The completeness of the calculi is an immediate consequence of the admissibility of cut. We show that one can derive a decision procedure from the cut-free calculi, providing a constructive proof of decidability of the logics considered. As usual, we obtain a terminating proof search mechanism by controlling the backward application of some critical rules. By estimating the size of the finite derivations of a given sequent, we also obtain a polynomial space complexity bound for these logics.

We have implemented the calculi SeqS' in Prolog following the line of CondLean: our theorem prover is inspired to the "lean" methodology, whose basic idea is to write short programs and exploit the power of Prolog's engine as much as possible. The implementation offer significantly better performances with respect to those of CondLean, allowing us to conclude that the calculi SeqS' can be considered a first, plausible solution to the problem of reasoning in conditional logics with CEM.

The plan of the paper is as follows. In Section 2 we introduce Conditional Logics with Conditional Excluded Middle. In Section 3 we present SeqS', the novel labelled sequent calculi, by emphasizing the differences with SeqS. In Section 4 we describe a Prolog implementation of SeqS', then we conclude in Section 6 with some experimental results witnessing that its performance are better than those of CondLean. and with some pointers to future works.

2. Conditional Logics with Conditional Excluded Middle

In this section we briefly present propositional conditional logics with CEM.

A propositional conditional language $\mathcal{L}$ contains: (i) a set of propositional variables ATM; (ii) the constants $\bot$ and $\top$; (iii) a set of connectives $\neg$ (unary), $\wedge$, $\vee$, $\rightarrow$, $>$ (binary). Formulas of $\mathcal{L}$ as follows:

- $\bot$, $\top$, and the propositional variables of ATM are *atomic formulas*;
- if A and B are formulas, $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$ and $A > B$ are *complex formulas.*

We define the *selection function semantics* as follows: given a non-empty set of possible worlds $\mathcal{W}$, the selection function f selects, for a world w and a formula A, the set of worlds of $\mathcal{W}$

which are *closer* to w given the information A. A conditional formula $A > B$ holds in a world w if the formula B holds in *all the worlds selected by f for w and A*.

Definition 1 (Selection function semantics). A model is a triple $\mathcal{M} = \langle \mathcal{W}, f, [\,] \rangle$ where:

- $\mathcal{W}$ is a non empty set of *worlds*;
- f is the *selection function*
$$f : \mathcal{W} \times 2^{\mathcal{W}} \longrightarrow 2^{\mathcal{W}}$$
satisfying the condition for *conditional excluded middle*:
$$\mid f(w, [A]) \mid \leq 1$$
- $[\,]$ is the *evaluation function*, which assigns to an atom $P \in ATM$ the set of worlds where P is true, and is extended to the other formulas as follows:
  - $[\bot] = \emptyset$;
  - $[\top] = \mathcal{W}$;
  - $[\neg A] = \mathcal{W} \setminus [A]$;
  - $[A \wedge B] = [A] \cap [B]$;
  - $[A \vee B] = [A] \cup [B]$;
  - $[A \rightarrow B] = (\mathcal{W} \backslash [A]) \cup [B]$;
  - $[A > B] = \{w \in \mathcal{W} \mid f(w, [A]) \subseteq [B]\}$.

It is worth noticing that we have defined f taking $[A]$ rather than A (i.e. $f(w,[A])$ rather than $f(w,A)$) as an argument; this is equivalent to define f on formulas, i.e. $f(w,A)$ but imposing that if $[A]$=$[A^{'}]$ in the model, then $f(w, A)$=$f(w, A^{'})$. This condition is called *normality*.

The semantics above characterizes the basic conditional system we consider, called CK+CEM. An axiomatization of this system is given by:

- any axiomatization of classical propositional calculus;
- (CEM) $\quad (A > B) \vee (A > \neg B)$
- (Modus Ponens) $\quad \dfrac{A \quad A \rightarrow B}{B}$
- (RCEA) $\quad \dfrac{A \leftrightarrow B}{(A > C) \leftrightarrow (B > C)}$
- (RCK) $\quad \dfrac{(A_1 \wedge \cdots \wedge A_n) \rightarrow B}{(C > A_1 \wedge \cdots \wedge C > A_n) \rightarrow (C > B)}$

As for modal logics, we can consider extensions of CK+CEM by assuming further properties on the selection function. We consider the following ones:

Logic	**Axiom**	**Model condition**
ID	$A > A$	$f(w, [A]) \subseteq [A]$
CS	$(A \wedge B) \rightarrow (A > B)$	$w \in [A] \rightarrow f(w, [A]) \subseteq \{w\}$

The above axiomatization is complete with respect to the semantics [2].

3. A Labelled Sequent Calculus for Conditional Logics with CEM

We introduce SeqS', a sequent calculus for the conditional systems with CEM. The calculi make use of labels to represent possible worlds. We consider a language $\mathcal{L}$ and a denumerable alphabet of labels $\mathcal{A}$, whose elements are denoted by *x, y, z, ...* There are two kinds of labelled formulas:

- *world formulas*, denoted by *x: A*, where $x \in \mathcal{A}$ and $A \in \mathcal{L}$, used to represent that A holds in a world *x*;
- *transition formulas*, denoted by $x \xrightarrow{A} y$, where $x, y \in \mathcal{A}$ and $A \in \mathcal{L}$. A transition formula $x \xrightarrow{A} y$ represents that $y \in f(x, [A])$.

A *sequent* is a pair $\langle \Gamma, \Delta \rangle$, usually denoted with $\Gamma \vdash \Delta$, where Γ and Δ are multisets of labelled formulas. The intuitive meaning of $\Gamma \vdash \Delta$ is: every model that satisfies all labelled formulas of Γ in the respective worlds (specified by the labels) satisfies at least one of the labelled formulas of Δ (in those worlds). Formally, given a model $\mathcal{M} = \langle \mathcal{W}, f, [\,] \rangle$ for $\mathcal{L}$, and a label alphabet $\mathcal{A}$, we consider any *mapping* $I : \mathcal{A} \to \mathcal{W}$. Let F be a labelled formula, we define $\mathcal{M} \models_I F$ as follows:

- $\mathcal{M} \models_I x\colon A$ if and only if $I(x) \in [A]$
- $\mathcal{M} \models_I x \xrightarrow{A} y$ if and only if $I(y) \in f(I(x), [A])$

We say that $\Gamma \vdash \Delta$ is *valid* in $\mathcal{M}$ if for every mapping $I : \mathcal{A} \to \mathcal{W}$, if $\mathcal{M} \models_I F$ for every $F \in \Gamma$, then $\mathcal{M} \models_I G$ for some $G \in \Delta$. We say that $\Gamma \vdash \Delta$ is valid in a system, either the basic CK+CEM or any extension of it, if it is valid in every $\mathcal{M}$ satisfying the specific conditions for that system.

The calculi SeqS' are shown in Figure 1. We say that a sequent $\Gamma \vdash \Delta$ is *derivable* if it admits a derivation in SeqS', i.e. a proof tree, obtained by applying backwards the rules of the calculi, having $\Gamma \vdash \Delta$ as a root and whose leaves are all instances of (AX). As usual, the idea is as follows: in order to prove that a formula F is valid in a conditional logic, then one has to check whether the sequent $\vdash x : F$ is derivable in SeqS', i.e. if there is a derivation, obtained by applying backwards the rules, having $\vdash x : F$ as a root.

As a difference with the starting point of this work, namely the sequent calculi SeqS introduced in [19], the calculi SeqS' deal with the CEM condition by means of a second rule whose principal formula is a conditional $A > B$ on the right-hand side of a sequent, in addition to the "standard" one already belonging to the original calculus. The novel rule, called $(CEM^{>})$, is as follows:

$$\frac{\Gamma \vdash \Delta, x \xrightarrow{A} y \qquad \Gamma \vdash \Delta, y : B}{\Gamma \vdash \Delta, x : A > B}\ (CEM^{>})$$

This rule replaces the following rule (CEM) of SeqS:

$$\frac{\Gamma, x \xrightarrow{A} y \vdash \Delta, x \xrightarrow{A} z \qquad (\Gamma, x \xrightarrow{A} y \vdash \Delta)[y/u, z/u]}{\Gamma, x \xrightarrow{A} y \vdash \Delta}\ (CEM)$$

where $\Sigma[x/u]$ is used to denote the multiset obtained from Σ by replacing the label x by u wherever it occurs, and where it holds that $y \neq z$ and $u \notin \Gamma, \Delta$. The basic idea underlying the

CK+CEM

$$(AX)\ \Gamma, x : P \vdash \Delta, x : P \quad (P \in ATM) \qquad (AX)\ \Gamma, x : \bot \vdash \Delta \qquad (AX)\ \Gamma \vdash \Delta, x : \top$$

$$(\neg\mathbf{L})\frac{\Gamma \vdash \Delta, x : A}{\Gamma, x : \neg A \vdash \Delta} \qquad (\neg\mathbf{R})\frac{\Gamma, x : A \vdash \Delta}{\Gamma \vdash \Delta, x : \neg A} \qquad (\vee\mathbf{L})\frac{\Gamma, x : A \vdash \Delta \qquad \Gamma, x : B \vdash \Delta}{\Gamma, x : A \vee B \vdash \Delta}$$

$$(\vee\mathbf{R})\frac{\Gamma \vdash \Delta, x : A, x : B}{\Gamma \vdash \Delta, x : A \vee B} \qquad (\wedge\mathbf{L})\frac{\Gamma, x : A, x : B \vdash \Delta}{\Gamma, x : A \wedge B \vdash \Delta} \qquad (\wedge\mathbf{R})\frac{\Gamma \vdash \Delta, x : A \qquad \Gamma \vdash \Delta, x : B}{\Gamma \vdash \Delta, x : A \wedge B}$$

$$(\rightarrow\mathbf{L})\frac{\Gamma \vdash \Delta, x : A \qquad \Gamma, x : B \vdash \Delta}{\Gamma, x : A \rightarrow B \vdash \Delta} \qquad (>\mathbf{L})\frac{\Gamma, x : A > B \vdash \Delta, x \xrightarrow{A} y \qquad \Gamma, x : A > B, y : B \vdash \Delta}{\Gamma, x : A > B \vdash \Delta}$$

$$(\rightarrow\mathbf{R})\frac{\Gamma, x : A \vdash \Delta, x : B}{\Gamma \vdash \Delta, x : A \rightarrow B} \qquad (>\mathbf{R})\frac{\Gamma, x \xrightarrow{A} y \vdash \Delta, y : B, x : A > B}{\Gamma \vdash \Delta, x : A > B}(y \notin \Gamma, \Delta)$$

$$(CEM^{>})\frac{\Gamma \vdash \Delta, x \xrightarrow{A} y \qquad \Gamma \vdash \Delta, y : B}{\Gamma \vdash \Delta, x : A > B} \qquad (\mathbf{EQ})\frac{u : A \vdash u : B \qquad u : B \vdash u : A}{\Gamma, x \xrightarrow{A} y \vdash \Delta, x \xrightarrow{B} y}$$

$$(\mathbf{CS})\frac{\Gamma, x \xrightarrow{A} y \vdash \Delta, x : A \qquad \Gamma[x/u, y/u], u \xrightarrow{A} u \vdash \Delta[x/u, y/u]}{\Gamma, x \xrightarrow{A} y \vdash \Delta}\ (x \neq y, u \notin \Gamma, \Delta) \qquad (\mathbf{ID})\frac{\Gamma, x \xrightarrow{A} y, y : A \vdash \Delta}{\Gamma, x \xrightarrow{A} y \vdash \Delta}$$

Extensions

Figure 1: Rules of sequent calculi SeqS'

.

new formulation is to generate a new label when dealing with a conditional $x : A > B$ on the right-hand side of a sequent only one time, in order to generate a single world belonging to the selection function of the world represented by x for A, satisfying the semantic condition of having at most one such a world. As an example, Figure 2 shows a derivation of an instance of the characterizing axiom (CEM).

It is easy to observe that the rule $(> \mathbf{R})$ is first applied to $A > B$, introducing the new label y, representing the world selected by the selection function. Then, when the other conditional $A > \neg B$ is taken into account, the rule $(> \mathbf{R})$ is no longer applied, however the new rule $(CEM^{>})$ is applied by selecting the world represented by y as the only one belonging to the "most similar" worlds to the one represented by x given the formula A.

The following basic structural properties hold for all the calculi SeqS' (proofs are similar to those in [19] and omitted to save space.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(AX)}
\UnaryInfC{$x \xrightarrow{A} y \vdash x \xrightarrow{A} y, x : A > B, y : B$}
\AxiomC{}
\RightLabel{(AX)}
\UnaryInfC{$x \xrightarrow{A} y, y : B \vdash x : A > B, y : B$}
\RightLabel{$(\neg \mathbf{R})$}
\UnaryInfC{$x \xrightarrow{A} y \vdash x : A > B, y : B, y : \neg B$}
\RightLabel{$(CEM^{>})$}
\BinaryInfC{$x \xrightarrow{A} y \vdash x : A > B, x : A > \neg B, y : B$}
\RightLabel{$(> \mathbf{R})$}
\UnaryInfC{$\vdash x : A > B, x : A > \neg B$}
\RightLabel{$(\vee \mathbf{R})$}
\UnaryInfC{$\vdash x : (A > B) \vee (A > \neg B)$}
\end{prooftree}
$$

Figure 2: A derivation of CEM in SeqS'.

Theorem 1 (Height-preserving admissibility of weakening). *If $\Gamma \vdash \Delta$ is derivable in SeqS' with a derivation whose height is h, then also are $\Gamma \vdash \Delta, F$ and $\Gamma, F \vdash \Delta$, with proofs of height $h_1 \leq h$ and $h_2 \leq h$, respectively, where F is any labelled formula.*

Theorem 2 (Height-preserving invertibility of the rules). *If $\Gamma \vdash \Delta$ is derivable in SeqS' with a derivation whose height is h, and $\Gamma \vdash \Delta$ is an instance of the conclusion of a rule R of SeqS', then also $\Gamma' \vdash \Delta'$, where $\Gamma' \vdash \Delta'$ is an instance of one of the premises of R, is derivable in SeqS' with a proof of height $h' \leq h$.*

Theorem 3 (Height-preserving admissibility of contraction). *If $\Gamma \vdash \Delta, F, F$, where F is any labelled formula, is derivable in SeqS' with a derivation whose height is h, then also $\Gamma \vdash \Delta, F$ is derivable in SeqS' with a proof of height $h' \leq h$. If $\Gamma, F, F \vdash \Delta$, where F is any labelled formula, is derivable in SeqS' with a derivation whose height is h, then also $\Gamma, F \vdash \Delta$ is derivable in SeqS' with a proof of height $h' \leq h$.*

The calculi SeqS' are sound and complete for all the systems considered, namely the basic system CK+CEM, as well as the three extensions with ID, CS, and both CS and ID:

Theorem 4 (Soundness and completeness). *Given a conditional formula F, it is valid in a conditional logic with conditional excluded middle if and only if it is derivable in the corresponding calculus of SeqS', that it to say $\models F$ if and only if $\vdash x : F$ is derivable in SeqS'.*

Proof. For the soundness, we have to prove that, if a sequent $\Gamma \vdash \Delta$ is derivable, then the sequent is valid. This can be done by induction on the height of the derivation of $\Gamma \vdash \Delta$. The basic cases are those corresponding to derivations of height 0, that is to say instances of (AX). It is easy to see that, in all these cases, $\Gamma \vdash \Delta$ is a valid sequent. As an example, consider $\Gamma, x : P \vdash \Delta, x : P$: consider every model $\mathcal{M}$ and every mapping I satisfying all formulas in the left-hand side of the sequent, then also $x : P$. This means that $I(x) \in [P]$, but then we have that $\mathcal{M}$ satisfies via I at least a formula in the right-hand side of the sequent, the same $x : P$. For the inductive step, we proceed by considering each rule of the calculi SeqS' in order to check that, if the premise(s) is (are) valid sequent(s), to which we can apply the inductive hypothesis, so is the conclusion. Due to space limitations, we only present the case of the new rule $(CEM^{>})$, for the other rules the proof is similar to the one of SeqS in [19]. Let the considered proof ended as:

$$\frac{(1)\ \Gamma \vdash \Delta, x \xrightarrow{A} y \qquad (2)\ \Gamma \vdash \Delta, y : B}{(3)\ \Gamma \vdash \Delta, x : A > B}\ (CEM^{>})$$

By inductive hypothesis, both (1) and (2) are valid. By absurd, suppose (3) is not, that is to say there exists a model $\mathcal{M}$ and a mapping I satisfying all formulas in Γ but falsifying all formulas in Δ as well as $x : A > B$. Since (1) is valid, since $\mathcal{M}$ and I falsifies all formulas in Δ, necessarily we have that $\mathcal{M} \models_I x \xrightarrow{A} y$, that is to say $I(y) \in f(I(x), [A])$. By the CEM semantic condition, it follows that $(*)\ f(I(x), [A]) = \{I(y)\}$. Analogously, by the validity of (2) we have that $\mathcal{M} \models_I y : B$. If $\mathcal{M} \not\models_I x : A > B$ in (3), there exists a world w such that $w \in f(I(x), [A])$ and $w \notin [B]$, however, since $(*)$, we have that $I(y) = w$, against the validity of (2), and we are done.

The completeness is an easy consequence of the admissibility of the *cut* rule:

$$\frac{\Gamma \vdash \Delta, F \quad F, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}\ (cut)$$

where F is any labelled formula. As usual, the proof proceeds by a double induction over the complexity of the cut formula and the sum of the heights of the derivations of the two premises of cut, in the sense that we replace one cut by one or several cuts on formulas of smaller complexity, or on sequents derived by shorter derivations. We only show one of the paradigmatic cases involving the novel rule $(CEM^{>})$, namely the case in which the cut formula is the principal formulas in both the premises of (cut), and the rules applied to it are $(CEM^{>})$ and $(> \mathbf{L})$. The situation is as follows:

$$\frac{\dfrac{(1)\Gamma \vdash \Delta, x \xrightarrow{A} y \quad (2)\Gamma \vdash \Delta, y : B}{(5)\Gamma \vdash \Delta, x : A > B}\ (CEM^{>}) \qquad \dfrac{(3)\Gamma, x : A > B \vdash \Delta, x \xrightarrow{A} y \quad (4)\Gamma, x : A > B, y : B \vdash \Delta}{(6)\Gamma, x : A > B \vdash \Delta}\ (> \mathbf{L})}{\Gamma \vdash \Delta}\ (cut)$$

Since weakening is height-preserving admissible, we can obtain a proof (with a derivation of at most the same height of (5)) for $(5')\ \Gamma \vdash \Delta, x : A > B, y : B$. By inductive hypothesis on the height of the derivations, we can cut (4) and $(5')$, obtaining a derivation of $(7)\ \Gamma, y : B \vdash \Delta$. We can then apply the inductive hypothesis on the complexity of the cut formula to cut (2) and (7), and we are done with a derivation of $\Gamma \vdash \Delta$. The remaining cases are similar to those in [19] and left to the reader.

With the rule (cut) at hand, we show that if a formula F is valid in a conditional logic with CEM, then $\vdash x : F$ is derivable in SeqS'. We proceed by induction on the complexity of the formulas, therefore we show that the axioms are derivable and that the set of derivable formulas is closed under (Modus Ponens), (RCEA), and (RCK). A derivation of axioms **(ID)** and **(CS)** can be obtained as in SeqS [19]. A derivation of (CEM) is provided in Figure 2. For (Modus Ponens), suppose that $\vdash x : A \rightarrow B$ and $\vdash x : A$ are derivable. We easily have that $x : A \rightarrow B, x : A \vdash x : B$ is derivable too. Since cut is admissible, by two cuts we obtain

$\vdash x : B$:

$$\dfrac{\dfrac{x : A \to B, x : A \vdash x : B \qquad \vdash x : A \to B}{x : A \vdash x : B}\,(cut) \qquad \vdash x : A}{\vdash x : B}\,(cut)$$

For (RCEA), we have to show that if $A \leftrightarrow B$ is derivable, then also $(A > C) \leftrightarrow (B > C)$ is so. The formula $A \leftrightarrow B$ is an abbreviation for $(A \to B) \wedge (B \to A)$. Suppose that $\vdash x : (A \to B) \wedge (B \to A)$ is derivable, then also $x : A \vdash x : B$ and $x : B \vdash x : A$ are derivable since rules are height-preserving invertible. We can derive $x : A > C \vdash x : B > C$ as follows:

$$\dfrac{\dfrac{x : A \vdash x : B \qquad x : B \vdash x : A}{x : A > C, x \xrightarrow{B} y \vdash x \xrightarrow{A} y, y : C}\,(\mathbf{EQ}) \qquad x : A > C, x \xrightarrow{B} y, y : C \vdash y : C}{\dfrac{x \xrightarrow{B} y, x : A > C \vdash y : C}{x : A > C \vdash x : B > C}\,(> \mathbf{R})}\,(> \mathbf{L})$$

The other half is symmetric. For (RCK), suppose that (1) $\vdash x : B_1 \wedge B_2 \cdots \wedge B_n \to C$ is derivable, by the height-preserving invertibility of the rules also $y : B_1, \ldots, y : B_n \vdash y : C$ is derivable. We obtain the following derivation:

$$\dfrac{x \xrightarrow{A} y \vdash x \xrightarrow{A} y \qquad x : A > B_1, y : B_1, \ldots, y : B_n \vdash y : C}{x \xrightarrow{A} y, x : A > B_1, y : B_1, \ldots, y : B_{n-1} \vdash y : C}\,(\Rightarrow \mathbf{L})$$

$$\vdots$$

$$\dfrac{x \xrightarrow{A} y \vdash x \xrightarrow{A} y \qquad x \xrightarrow{A} y, x : A > B_1, \ldots, x : A > B_n, y : B_1 \vdash y : C}{\dfrac{x \xrightarrow{A} y, x : A > B_1, \ldots, x : A > B_n \vdash y : C}{x : A > B_1, \ldots, x : A > B_n \vdash x : A > C}\,(> \mathbf{R})}\,(> \mathbf{L})$$

■

The presence of labels and of the rules ($>$ **L**), $(\mathbf{ID})$, and $(\mathbf{CS})$, which increase the complexity of the sequent in a backward proof search, is a potential cause of a non-terminating proof search. However, with a similar argument to the one proposed in [19], we can define a procedure that can apply such rules in a controlled way and introducing a finite number of labels, ensuring termination. Intuitively, it can be shown that it is useless to apply ($>$ **L**) on $x : A > B$ by introducing (looking backward) the same transition formula $x \xrightarrow{A} y$ more than once in each branch of a proof tree. Similarly, it is useless to apply $(\mathbf{ID})$ or $(\mathbf{CS})$on the same transition $x \xrightarrow{A} y$ more than once in a backward proof search in each branch of a derivation. This leads to the decidability of the given logics:

Theorem 5 (Decidability). *Conditional logics CK+CEM, CK+CEM+ID, CK+CEM+CS, and CK+CEM+ID+CS are decidable.*

We can show that provability in all the conditional logics with CEM considered is decidable in $O(n^2 \ \log n)$ space, the proof is essentially the same as in [19] and can be omitted in order to save space.

4. A Theorem Prover for Conditional Logics with CEM

We have implemented the calculi SeqS' introduced in the previous section (https://gitlab2.educ.di.unito.it/pozzato/condlean4) in order to show that such a calculus can be the base for efficient theorem proving for conditional logics with conditional excluded middle. In order to provide a safe and direct comparison with CondLean [20, 21], as far as we know, the only theorem prover for these logics, we have followed the so-called "lean" methodology, introduced by Beckert and Posegga in the middle of the 90s [22, 23, 24]. Beckert and Posegga have proposed a very elegant and extremely efficient first-order theorem prover, called leanT^AP, consisting of only five Prolog clauses. The basic idea of the "lean" methodology is "to achieve maximal efficiency from minimal means" [22] by writing short programs and exploiting the power of Prolog's engine as much as possible.

We implement each component of a sequent by a list of formulas, partitioned into three sub-lists: atomic formulas, transitions and complex formulas. Atomic and complex formulas are implemented by a Prolog list of the form `[x,a]`, where `x` is a Prolog constant and `a` is a formula. A transition formula $x \xrightarrow{A} y$ is implemented by a Prolog list of the form `[x,a,y]`. Labels are implemented by Prolog constants. The sequent calculi are implemented by the predicate

prove(Cond, Gamma, Delta, Labels, Tree)

which succeeds if and only if $\Gamma \vdash \Delta$ is derivable in SeqS, where `Gamma` and `Delta` are the lists implementing the multisets Γ and Δ, respectively and `Labels` is the list of labels introduced in that branch. `Cond` is a list of pairs of kind $[F, \mathit{Used}]$, where F is a conditional formula `[X,A => B]` and *Used* is a list of transitions $[[X, A_1, Y_1], \ \ldots, [X, A_n, Y_n]]$ such that ($>$ L) has already been applied to $x : A > B$ by using transitions $x \xrightarrow{A_i} y_i$. The list `Cond` is used in order to ensure the termination of the proof search, by applying the restrictions described in the previous section in order to avoid useless applications of the rules. Similar mechanisms are adopted for extensions of the basic system CK+CEM, in order to control the applications of rules (ID) and (CS). `Tree` is an output term: if the proof search succeeds, it matches an implementation of the derivation found by the theorem prover.

Each clause of the `prove` predicate implements one axiom or rule of SeqS'. The theorem prover proceeds as follows. First of all, if $\Gamma \vdash \Delta$ is an axiom, then the goal will succeed immediately by using the clauses for the axioms. If it is not, then the first applicable rule is chosen. The ordering of the clauses is such that the application of the branching rules is postponed as much as possible. Concerning the rules for $>$ on the right-hand side of a sequent, the rule ($>$ R), which introduces a new label in a backward proof search, is first applied to a sequent of the form $\Gamma \vdash \Delta, x : A > B$. If this does not lead to a derivation, the new rule for CEM is then applied.

As an example, the clause for the axiom checking whether the same atomic formula occurs in both the left and the right hand side of a sequent is implemented as follows:

prove(_,[LitGamma,_,_],[LitDelta,_,_],_) : -

```
  member(F,LitGamma),member(F,LitDelta),!.
```

As another example, here is the clause implementing (> L):

prove(Cond,[LitGamma,TransGamma,ComplexGamma],
[LitDelta,TransDelta,ComplexDelta], Labels) : -

```
  member([X,A => B],ComplexGamma),
  select([[X,A => B],Used],Cond,TempCond),
  member([X,C,Y],TransGamma),
  \+member([X,C,Y],Used),!,
  put([Y,B],LitGamma,ComplexGamma,NewLitGamma,
      NewComplexGamma),
  prove([[[X, A => B],[[X,C,Y] | Used]] | TempCond],
    [LitGamma,TransGamma,ComplexGamma],
    [LitDelta,[[X,A,Y]|TransDelta],ComplexDelta],Labels),
  prove([[[X, A => B],[[X,C,Y] | Used]] | TempCond],
    [NewLitGamma,TransGamma,NewComplexGamma],
    [LitDelta,TransDelta,ComplexDelta],Labels).
```

The predicate `put` is used to put `[Y,B]` in the proper sub-list of the antecedent. The two recursive calls to `prove` implement the proof search on the two premises of the rule.

As a further example, here is the code of the novel rule $(CEM^{>})$:

prove(Cond,[LitGamma,TransGamma,ComplexGamma],
[LitDelta,TransDelta,ComplexDelta], Labels) : -

```
  select([X,A => B],ComplexDelta,ResComplexDelta),!,
  member([X,_,Y],TransGamma),
  put([Y,B],LitDelta,ResComplexDelta,NewLitDelta,NewComplexDelta),
  prove(Cond,[LitGamma,TransGamma,ComplexGamma],
      [LitDelta,[[X,A,Y] | TransDelta],ComplexDelta], Labels),
  prove(Cond,[LitGamma,TransGamma,ComplexGamma],
      [NewLitDelta,TransDelta,NewComplexDelta], Labels).
```

In order to search a derivation of a sequent $\Gamma \vdash \Delta$, the theorem prover proceeds as follows. First, if $\Gamma \vdash \Delta$ is an axiom, the goal will succeed immediately by using the clauses for the axioms. If it is not, then the first applicable rule is chosen, e.g. if `ComplexDelta` contains a formula `[X,A -> B]`, then the clause for ($\rightarrow$ R) rule is used, invoking `prove` on the unique premise of ($\rightarrow$ R). The prover proceeds in a similar way for the other rules. The ordering of the clauses is such that the application of the branching rules is postponed as much as possible.

In order to check whether a formula is valid in one of the considered system, one has just to invoke the following auxiliary predicate:

pr(Formula)

which wraps the `prove` predicate by a suitable initialization of its parameters.

The theorem prover is available for free download at https://gitlab2.educ.di.unito.it/pozzato/condlean4, where one can also find an updated version of CondLean in order to compute the statistics described in the next section.

5. Statistics

We have tested both CondLean and our theorem prover over

1. a set of randomly generated formulas, either valid or not
2. a set of formulas holding only in systems with CEM

obtaining the following results:

1. over randomly generated formulas, we have observed an improvement of the performances of CondLean of $48,27\%$.
2. over a set of valid formulas we are able to improve the performances of CondLean of $20,57\%$. As an example, running both the provers over the formula

   $$(A > (B_1 \vee \dots B_5)) > ((A > B_1) \vee \dots \vee (A > B_5))$$

   our theorem prover is able to build a derivation in 94 ms, against the 266 ms needed by CondLean.

We are currently testing the performances of our implementation over the extensions with ID and CS and we are developing a graphical interface for the prover and we are also providing Prolog files that will allow the user to reproduce a detailed comparison between the two systems in a completely automated way.

The performance of the proposed theorem prover are promising, especially concerning all cases in which it has to answer *no* for a not valid formula: this is justified by the fact that CondLean has to make a great effort in order to explore the whole space of alternative choices in label substitution, operation needed in order to conclude that no derivation exist.

6. Conclusions and Future Works

In this work we have introduced labelled sequent calculi for conditional logics with the axiom of conditional excluded middle (CEM), as well as all the extensions with axioms ID and CS. Our calculi revise those introduced in [19], where a modular labelled sequent calculus SeqS has been introduced for several conditional logics, including those with CEM. We have provided alternative calculi, where the original rule for CEM, based on an expensive mechanism of label substitution, has been replaced by a novel and "standard" rule, called $(CEM^{>})$ specifically tailored for handling conditional formulas $A > B$ in these systems.

We have also implemented the prosed calculi in order to obtain an empirical witness of the fact that our solution improves the one in [19]. We have compared the performances of our theorem prover with those of CondLean, a Prolog implementation of the calculi SeqS. Our implementation is inspired to the "lean" methodology and, in order to focus on CEM, it adopts all the choices of CondLean, essentially just by replacing the rule for conditional excluded middle with a clause implementing the novel $(CEM^{>})$.

In future work we plan to extend the calculi and the implementation to other conditional logics with conditional excluded middle. In particular, our main objective is to include extensions with the axiom MP of conditional modus ponens:

$$(A > B) \rightarrow (A \rightarrow B),$$

whose selection functions must respect the following condition:

$$\text{if } w \in [A], \text{ then } w \in f(w, [A]).$$

This system, as well as its extensions with ID and CS, is not handled by CondLean, since [19] does not show that (cut) is admissible also for them in the calculi SeqS.

Moreover, we aim at implementing a "concrete" theorem prover, starting from the one proposed in this work, implementing state of the art heuristics, data structures and suitable refinements. As already mentioned, we are currently working on extending the set of formulas used in order to obtain further statistics, with the objective of comparing the performances of the proposed theorem prover with those of CondLean.

Acknowledgments

This work has been partially supported by the INdAM - GNCS Project cod. CUP_E55F22000270001 "LESLIE: LogichE non-claSsiche per tooL Intelligenti ed Explainable".

References

[1] D. Lewis, Counterfactuals, Basil Blackwell Ltd (1973).

[2] D. Nute, Topics in Conditional Logic, Reidel, Dordrecht, 1980.

[3] R. Stalnaker, A theory of conditionals, in: N. Rescher (Ed.), Studies in Logical Theory, Blackwell, 1968, pp. 98–112.

[4] B. F. Chellas, Basic conditional logics, Journal of Philosophical Logic 4 (1975) 133–153.

[5] J. P. Burgess, Quick completeness proofs for some logics of conditionals, Notre Dame Journal of Formal Logic 22 (1981) 76–84.

[6] S. Kraus, D. Lehmann, M. Magidor, Nonmonotonic reasoning, preferential models and cumulative logics, Artificial Intelligence 44 (1990) 167–207.

[7] J. P. Delgrande, A first-order conditional logic for prototypical properties, Artificial Intelligence 33 (1987) 105–130.

[8] N. Friedman, J. Y. Halpern, Plausibility measures and default reasoning, Journal of the ACM 48 (2001) 648–685.

[9] L. Giordano, V. Gliozzi, N. Olivetti, G. L. Pozzato, Analytic tableaux for KLM preferential and cumulative logics, in: G. Sutcliffe, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings, volume 3835 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 666–681. URL: https://doi.org/10.1007/11591191_46. doi:10.1007/11591191_46.

[10] G. Grahne, Updates and counterfactuals, Journal of Logic and Computation 8 (1998) 87–117.

[11] L. Giordano, V. Gliozzi, N. Olivetti, Weak agm postulates and strong ramsey test: a logical formalization, Artificial Intelligence 168 (2005) 1–37.

[12] L. Giordano, V. Gliozzi, N. Olivetti, Iterated belief revision and conditional logic, Studia Logica 70 (2002) 23–47.

[13] H. Katsuno, A. O. Mendelzon, On the difference between updating a knowledge base and revising it, in: J. F. Allen, R. Fikes, E. Sandewall (Eds.), Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91). Cambridge, MA, USA, April 22-25, 1991, Morgan Kaufmann, 1991, pp. 387–394.

[14] D. M. Gabbay, L. Giordano, A. Martelli, N. Olivetti, M. L. Sapino, Conditional reasoning in logic programming, Journal of Logic Programming 44 (2000) 37–74.

[15] C. B. Schwind, Causality in action theories, Electronic Transactions on Artificial Intelligence (ETAI) 3 (1999) 27–50.

[16] L. Giordano, C. Schwind, Conditional logic of actions and causation, Artificial Intelligence 157 (2004) 239–279.

[17] V. Genovese, L. Giordano, V. Gliozzi, G. L. Pozzato, Logics in access control: a conditional approach, J. Log. Comput. 24 (2014) 705–762. URL: https://doi.org/10.1093/logcom/exs040. doi:10.1093/logcom/exs040.

[18] J. R. G. Williams, Defending conditional excluded middle, Noûs 44 (2010) 650–668. doi:10.1111/j.1468-0068.2010.00766.x.

[19] N. Olivetti, G. L. Pozzato, C. B. Schwind, A Sequent Calculus and a Theorem Prover for Standard Conditional Logics, ACM Transactions on Computational Logics (TOCL) 8 (2007).

[20] N. Olivetti, G. L. Pozzato, Condlean 3.0: Improving condlean for stronger conditional logics, in: B. Beckert (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings, volume 3702 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 328–332. URL: https://doi.org/10.1007/11554554_27. doi:10.1007/11554554_27.

[21] N. Olivetti, G. L. Pozzato, Condlean: A theorem prover for conditional logics, in: M. C. Mayer, F. Pirri (Eds.), Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2003, Rome, Italy, September 9-12, 2003. Proceedings, volume 2796 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 264–270. URL: https://doi.org/10.1007/978-3-540-45206-5_23. doi:10.1007/978-3-540-45206-5_23.

[22] B. Beckert, J. Posegga, leantap: Lean tableau-based deduction, Journal of Automated Reasoning 15 (1995) 339–358.

[23] B. Beckert, J. Posegga, Logic programming as a basis for lean automated deduction., Journal of Logic Programming 28 (1996) 231–236.

[24] M. Fitting, leantap revisited, Journal of Logic and Computation 8 (1998) 33–47.

Declarative Pattern Mining in Digital Forensics: Preliminary Results

Francesca Alessandra Lisi[1,*], Gioacchino Sterlicchio[1]

[1]University of Bari "Aldo Moro", Via E. Orabona 4, Bari, 70125, Italy

Abstract
This paper proposes the application of ASP-based sequential pattern mining techniques in the analysis of evidence collected according to the practice of digital forensics. In particular, it reports preliminary results concerning the analysis of anonymised mobile phone recordings, which highlight the sequences of events in a given time span.

Keywords
Sequential Pattern Mining, Answer Set Programming, Digital Forensics

1. Introduction

Digital Forensics (DF) is a branch of criminalistics which deals with the identification, acquisition, preservation, analysis and presentation of the information content of computer systems, or in general of digital devices, by means of specialized software, and according to specific regulations. In particular, the phase of *Evidence Analysis* involves examining and aggregating evidence about possible crimes and crime perpetrators collected from various electronic devices in order to reconstruct events, event sequences and scenarios related to a crime. Evidence Analysis results are made available to law enforcement, investigators, intelligence agencies, public prosecutors, lawyers and judges.

Unlike the phase of Identification, where the application of Machine Learning (ML) techniques can be useful for the analysis of big data, the phase of Evidence Analysis has particular requirements that make the use of techniques from *Knowledge Representation* (KR) and *Automated Reasoning* (AR) a much more promising approach, potentially becoming a breakthrough in the state-of-the-art. The ultimate goal of Evidence Analysis is indeed the formulation of verifiable evidence that can be rationally presented in a trial. Under this perspective, the results provided by ML classifiers or other types of "black box" AI systems do not have more value than human witness' suspicions and cannot be used as legal evidence. Logical methods provide a broad range of proof-based reasoning functionalities that can be implemented in a declarative framework where the problem specification and the computational program are closely aligned.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
*Corresponding author, affiliated to *Dipartimento di Informatica* and to *Centro Interdipartimentale di Logica e Applicazioni (CILA)* of the University of Bari.
FrancescaAlessandra.Lisi@uniba.it (F. A. Lisi); g.sterlicchio2@studenti.uniba.it (G. Sterlicchio)
http://www.di.uniba.it/~lisi/ (F. A. Lisi)
0000-0001-5414-5844 (F. A. Lisi)

This has the benefit that the correctness of the resulting systems can be formally verified. Moreover, recent research has led to new methods for visualising and explaining the results of computed answers (*e.g.*, based on argumentation schemes). So one can not only represent and solve relevant problems, but also provide tools to explain the conclusions (and their proofs) in a transparent, comprehensible and justified way. This approach to DF was first explored by Costantini *et al.* [1, 2], and subsequently adopted by the COST Action "Digital forensics: evidence analysis via intelligent systems and practices" (DigForASP)[1] which aims at promoting formal and verifiable AI methods and techniques for Evidence Analysis [3].

Pattern mining [4] is a class of data mining tasks that consist of extracting interesting structured patterns from a set of structured examples. These tasks encompass itemset mining, sequence mining and graph mining. The interestingness measure of a pattern is, in most of the algorithms, the number of its occurrences in the set of examples. Given a threshold k, interesting patterns are those that occur at least in k examples. In this case, the task is known as *frequent pattern mining* for which many algorithms have been proposed. Most of the efficient algorithmic solutions rely on an antimonotonicity property of the support: the larger the pattern, the fewer it occurs. Declarative pattern mining (DPM) aims at encoding pattern tasks in a declarative framework, and more specifically the frequent pattern mining tasks. Declarative pattern mining addressed the tasks of frequent itemset mining [5, 6], frequent sequential patterns [7, 8]. Different declarative frameworks have been explored: SAT [5], CP [9, 6], and ASP [8, 10]. We do not expect DPM to be competitive with dedicated algorithms, but to take advantage of the versatility of declarative frameworks to propose pattern mining tools that could exploit background knowledge during the mining process to extract less but meaningful patterns. In this paper we will consider the case of sequential patterns, which turn out to be promising as a support to the analysis of events and sequences of events in scenarios of interest to DF experts.

The paper is organized as follows. In Section 2 we provide the necessary preliminaries on ASP, sequential pattern mining and the ASP encoding used in our work. In Section 3 we describe the application to a typical DF problem: the analysis of mobile phone recordings. In Section 4 we report some preliminary experimental results. In Section 5 we conclude by commenting the ongoing work and by outlining some promising directions for research.

2. Preliminaries

2.1. Answer Set Programming

In the following we give a brief overview of the syntax and semantics of disjunctive logic programs in ASP. The reader can refer to, *e.g.*, [11] for a more extensive introduction to ASP.

Let U be a fixed countable set of (domain) elements, also called *constants*, upon which a total order $\prec$ is defined. An *atom* α is an expression $p(t_1, \ldots, t_n)$, where p is a predicate of arity $n \geq 0$ and each t_i is either a variable or an element from U (*i.e.*, the resulting language is function-free). An atom is *ground* if it is free of variables. We denote the set of all ground atoms

[1]DigForASP: https://digforasp.uca.es/

over U by B_U. A *(disjunctive) rule* r is of the form

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m$$

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, or a count expression of the form $\#count\{l : l_1, \ldots, l_i\} \bowtie u$, where l is an atom and l_j is a literal (*i.e.*, an atom which can be negated or not), $1 \geq j \geq i$, $\bowtie \in \{\leq, <, =, >, \geq\}$, and $u \in \mathbb{N}$. Moreover, "not" denotes *default negation*. The *head* of r is the set $head(r) = \{a_1, \ldots, a_n\}$ and the *body* of r is $body(r) = \{b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m\}$. Furthermore, we distinguish between $body^+(r) = \{b_1, \ldots, b_k\}$ and $body^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* if $n \leq 1$ and a *constraint* if $n = 0$. A rule r is *safe* if each variable in r occurs in $body^+(r)$. A rule r is *ground* if no variable occurs in r. A *fact* is a ground rule with $body(r) = \emptyset$ and $|head(r)| = 1$. An *(input) database* is a set of facts. A *program* is a finite set of rules. For a program Π and an input database D, we often write $\Pi(D)$ instead of $D \cup \Pi$. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground).

Given a program Π, let U_Π be the set of all constants appearing in Π. $Gr(\Pi)$ is the set of rules $r\sigma$ obtained by applying, to each rule $r \in \Pi$, all possible substitutions σ from the variables in r to elements of U_Π. For count-expressions, $\{l : l_1, \ldots, l_n\}$ denotes the set of all ground instantiations of l, governed through $l_1, \ldots, l_n$. An interpretation $I \subseteq B_U$ satisfies a ground rule r iff $head(r) \cap I \neq \emptyset$ whenever $body^+(r) \subseteq I$, $body^-(r) \cap I = \emptyset$, and for each contained count-expression, $N \bowtie u$ holds, where $N = |\{l | l_1, \ldots, l_n\}|$, $u \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$. A ground program Π is satisfied by I, if I satisfies each $r \in \Pi$. A non-ground rule r (resp., a program Π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\Pi)$). A subset-minimal set $I \subseteq B_U$ satisfying the *Gelfond-Lifschitz reduct* $\Pi^I = \{head(r) \leftarrow body^+(r) | I \cap body^-(r) = \emptyset, r \in Gr(\Pi)\}$ is called an *answer set* of Π. We denote the set of answer sets for a program Π by $AS(\Pi)$.

The tools used in this work are part of the Potassco[2] collection [12]. The main tool of the collection is the *clingo* ASP solver [13].

2.2. Sequential Pattern Mining

Our terminology on sequence mining follows the one in [7]. Throughout this article, $[n] = \{1, \ldots, n\}$ denotes the set of the first n positive integers.

Let Σ be the alphabet, *i.e.*, the set of items. An *itemset* $A = \{a_1, a_2, \ldots, a_m\} \subseteq \Sigma$ is a finite set of items. The size of A, denoted $|A|$, is m. A *sequence* s is of the form $s = \langle s_1 s_2 \ldots s_n \rangle$ where each s_i is an itemset, and n is the length of the sequence.

A *database* $\mathcal{D}$ is a multiset of sequences over Σ. A sequence $s = \langle s_1 \ldots s_m \rangle$ with $s_i \in \Sigma$ is contained in a sequence $t = \langle t_1 \ldots t_n \rangle$ with $m \leq n$, written $s \sqsubseteq t$, if $s_i \subseteq t_{e_i}$ for $1 \leq i \leq m$ and an increasing sequence $(e_1 \ldots e_m)$ of positive integers $e_i \in [n]$, called an *embedding* of s in t. For example, we have $\langle a(cd) \rangle \sqsubseteq \langle ab(cde) \rangle$ relative to embedding $(1, 3)$. (cd) denotes the itemset made of items c and d.

Given a database $\mathcal{D}$, the *cover* of a sequence p is the set of sequences in $\mathcal{D}$ that contain p: $cover(p, \mathcal{D}) = \{t \in D | p \sqsubseteq t\}$. The number of sequences in $\mathcal{D}$ containing p is called its *support*,

[2]Potassco: https://potassco.org/

Table 1
An example of sequence database $\mathcal{D}$.

Id	Sequence
1	$\langle d\ a\ b\ c\rangle$
2	$\langle a\ c\ b\ c\rangle$
3	$\langle a\ b\ c\rangle$
4	$\langle a\ b\ c\rangle$
5	$\langle a\ c\rangle$
6	$\langle b\rangle$
7	$\langle c\rangle$

that is, $supp(p, \mathcal{D}) = |cover(p, \mathcal{D})|$. For an integer k, the problem of *frequent sequence mining* is about discovering all sequences p such that $supp(p, \mathcal{D}) \geq k$. We often call p a (sequential) pattern, and k is also referred to as the (minimum) *support threshold*. For $k = 2$ we can see how $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle a\ b\rangle$, $\langle a\ c\rangle$, $\langle b\ c\rangle$ e $\langle a\ b\ c\rangle$ are common patterns in the database $\mathcal{D}$ reported in Table 1.

2.3. Mining sequential patterns with ASP

The sequence database $\mathcal{D}$ is represented in terms of ASP facts *seq(t, p, e)*, where the *seq* predicate says that an item *e* occurs at position *p* in a sequence *t*. For example, Listing 1 represents the seven sequences of Table 1 in ASP format.

```
seq(1,1,d). seq(1,2,a). seq(1,3,b).seq(1,4,c).
seq(2,1,a). seq(2,2,c). seq(2,3,b).seq(2,4,c).
seq(3,1,a). seq(3,2,b). seq(3,3,c).
seq(4,1,a). seq(4,2,b). seq(4,3,c).
seq(5,1,a). seq(5,2,c).
seq(6,1,b).
seq(7,1,c).
```

Listing 1: ASP encoding for the sequence database $\mathcal{D}$ reported in Table 1.

The ASP encoding for sequential pattern mining follows the principles outlined in [14] and [8]. In particular, there are two parameters to be defined: *maxlen* determines the maximum length of the patterns of interest and *th* specifies the minimum support threshold. The lower the value of *th* the more patterns will be extracted; the lower the *maxlen* parameter, the smaller the ground program will be. Therefore the parameters allow a tuning for the program efficiency. Also, each answer set comprises a single pattern of interest. More precisely, an answer set represents a frequent pattern $s = \langle s_i\rangle_{i\leq th\leq m}$ such that $1 \leq m \leq maxlen$ from atoms $pat(m, s_1), ..., pat(1, s_m)$. The first argument expresses the position of the object in increasing order, where *m* can vary, while *1* always indicates the first item in the pattern. For example the atoms *pat(1, a)*, *pat(2, b)* and *pat(3, c)* describe a frequent pattern $\langle a\ b\ c\rangle$ of the database in Table 1.

```
1   item(I) :- seq(_, _,I).
2
3   %sequential pattern generation
4   patpos(1).
5   0 { patpos(Ip+1) } 1 :- patpos(Ip), Ip<maxlen.
6   patlen(L) :- patpos(L), not patpos(L+1).
7
8   1 { pat(Ip,I): item(I) } 1 :- patpos(Ip).
9
10  %pattern embeddings
11  occ(T,1,Is) :- seq(T,Is,I), pat(1,I).
12  occ(T,Ip,Is) :- occ(T, Ip, Is-1), seq(T,Is,_).
13  occ(T,Ip,Is) :- occ(T, Ip-1, Is-1), seq(T,Is,I), pat(L,I).
14
15  %frequency constraint
16  seqlen(T,L) :- seq(T,L,_), not seq(T,L+1,_).
17  support(T) :- occF(T, L, LS), patlen(L), seqlen(T,LS).
18  :- { support(T) } < th.
```

Listing 2: Basic ASP encoding for sequential pattern mining [10].

Listing 2 reports the ASP program for sequential pattern mining according to [10]. Line 1 defines a new predicate that provides all items from the database. Lines 4 to 8 of the program encode the pattern generation. Lines 11 to 13 encode pattern embedding search. Finally, Lines 16 to 18 are dedicated to assess the pattern frequency constraint. For a thorough discussion of the program the reader can refer to [10].

3. Sequence Mining in Mobile Phone Records with ASP

During the investigation of a crime, it is common to analyze the communications of a particular suspect. Given that, nowadays the mobile phone or smartphone is an object owned by anyone, it can be useful for investigators to analyze the calls or messages exchanged. The telephone records are a set of data inserted in tables that contain the data relating to the external communications of the devices. In other words, the telephone records contain all the traces of communications relating to a specific user over a certain period of time, therefore they contain traces of telephone calls, SMS, and all the data traffic of the mobile phone.

Telephone records concern various pieces of information, such as:

- the telephone number of the caller
- the telephone number of the recipient
- the type of communication, *e.g.*: call, sms, missed call;
- the duration of the communication, indicated in minutes and seconds in the event of a call. On the other hand, in the case of text messages or missed calls, this value will be equal to 0 seconds.

In addition to the information indicated above, telephone records can report a series of additional information usually referred to mobile users and therefore related to communications via mobile phones.

Through a telephone records it is not possible to trace a series of important data such as the audio of calls sent or received, the list of SMS messages, the content of the e-mails received or sent, and the list of the web sites visited. In fact, through a telephone records it is possible to have a trace of the communication that has taken place but not to obtain its content. The telephone records can be requested by the Judicial Authority if it deems it useful to get hold of them in order to carry out investigations on the individual owner of the user.

Correctly analyzing the telephone records is essential to obtain useful data. Depending on the analysis, different types of information can be extracted. As a rule, the records are analyzed for comparing the geographical positions with respect to the declarations, and for reconstructing the network of contacts with reference to a single user in order to trace which conversations he/she has had with which people and at what times.

3.1. The DigForASP dataset

For our experiments we have considered a dataset which consists of the telephone records of four users from a real-word investigative case. The dataset has been made available by Prof. David Billard (University of Applied Sciences in Geneva) under NDA to DigForASP members for academic experimentation.

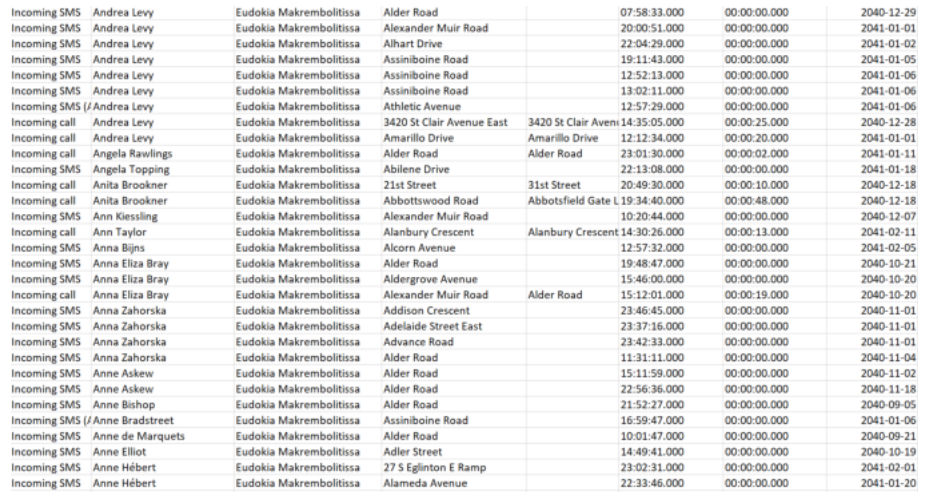
Each file in the dataset has the following schema:

- *Type*: what kind of operation the user has performed (*e.g.*, incoming/outgoing call or SMS);
- *Caller*: who makes the call or sends an SMS;
- *Callee*: who receives the call or SMS;
- *Street*: where the operation has taken place;
- *Time*: when the operation has taken place (ISO format[3] HH: MM: SS);
- *Duration*: how long the operation has been (ISO format HH: MM: SS);
- *Date*: when the operation has taken place (format: day, month, year).

The type of the operation is one of the following cases: "config", "gprs", "redirect", "out_sms(SUB_TYPE)", "in_sms(SUB_TYPE)", "out_call(SUB_TYPE)", "in_call(SUB_TYPE)". Subtypes are: "simple", "ack", "foreign".

The dataset has undergone the mandatory anonymization process for reasons of privacy and confidentiality. Therefore it does not contain data that allows tracing back to the real people involved in the investigative case. For instance, there is no phone number for the caller/callee but only a fictitious name. The names and the sizes (# rows) of the four files in the dataset are the following: Eudokia Makrembolitissa (8,783), Karen Cook McNally (20,894), Laila Lalami (12,689), and Lucy Delaney (8,480). An excerpt of the file containing the phone recordings of Eudokia Makrembolitissa is reported in Figure 1.

[3]Format to describe dates and times: https://en.wikipedia.org/wiki/ISO_8601



Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Alder Road		07:58:33.000	00:00:00.000	2040-12-29
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Alexander Muir Road		20:00:51.000	00:00:00.000	2041-01-01
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Alhart Drive		22:04:29.000	00:00:00.000	2041-01-02
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Assiniboine Road		19:11:43.000	00:00:00.000	2041-01-05
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Assiniboine Road		12:52:13.000	00:00:00.000	2041-01-06
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Assiniboine Road		13:02:11.000	00:00:00.000	2041-01-06
Incoming SMS (/	Andrea Levy	Eudokia Makrembolitissa	Athletic Avenue		12:57:29.000	00:00:00.000	2041-01-06
Incoming call	Andrea Levy	Eudokia Makrembolitissa	3420 St Clair Avenue East	3420 St Clair Aven	14:35:05.000	00:00:25.000	2040-12-28
Incoming call	Andrea Levy	Eudokia Makrembolitissa	Amarillo Drive	Amarillo Drive	12:12:34.000	00:00:20.000	2041-01-01
Incoming call	Angela Rawlings	Eudokia Makrembolitissa	Alder Road	Alder Road	23:01:30.000	00:00:02.000	2041-01-11
Incoming SMS	Angela Topping	Eudokia Makrembolitissa	Abilene Drive		22:13:08.000	00:00:00.000	2041-01-18
Incoming call	Anita Brookner	Eudokia Makrembolitissa	21st Street	31st Street	20:49:30.000	00:00:10.000	2040-12-18
Incoming call	Anita Brookner	Eudokia Makrembolitissa	Abbottswood Road	Abbotsfield Gate L	19:34:40.000	00:00:48.000	2040-12-18
Incoming SMS	Ann Kiessling	Eudokia Makrembolitissa	Alexander Muir Road		10:20:44.000	00:00:00.000	2040-12-07
Incoming call	Ann Taylor	Eudokia Makrembolitissa	Alanbury Crescent	Alanbury Crescent	14:30:26.000	00:00:13.000	2041-02-11
Incoming SMS	Anna Bijns	Eudokia Makrembolitissa	Alcorn Avenue		12:57:32.000	00:00:00.000	2041-02-05
Incoming SMS	Anna Eliza Bray	Eudokia Makrembolitissa	Alder Road		19:48:47.000	00:00:00.000	2040-10-21
Incoming SMS	Anna Eliza Bray	Eudokia Makrembolitissa	Aldergrove Avenue		15:46:00.000	00:00:00.000	2040-10-20
Incoming call	Anna Eliza Bray	Eudokia Makrembolitissa	Alexander Muir Road	Alder Road	15:12:01.000	00:00:19.000	2040-10-20
Incoming SMS	Anna Zahorska	Eudokia Makrembolitissa	Addison Crescent		23:46:45.000	00:00:00.000	2040-11-01
Incoming SMS	Anna Zahorska	Eudokia Makrembolitissa	Adelaide Street East		23:37:16.000	00:00:00.000	2040-11-01
Incoming SMS	Anna Zahorska	Eudokia Makrembolitissa	Advance Road		23:42:33.000	00:00:00.000	2040-11-01
Incoming SMS	Anna Zahorska	Eudokia Makrembolitissa	Alder Road		11:31:11.000	00:00:00.000	2040-11-04
Incoming SMS	Anne Askew	Eudokia Makrembolitissa	Alder Road		15:11:59.000	00:00:00.000	2040-11-02
Incoming SMS	Anne Askew	Eudokia Makrembolitissa	Alder Road		22:56:36.000	00:00:00.000	2040-11-18
Incoming SMS	Anne Bishop	Eudokia Makrembolitissa	Alder Road		21:52:27.000	00:00:00.000	2040-09-05
Incoming SMS (/	Anne Bradstreet	Eudokia Makrembolitissa	Assiniboine Road		16:59:47.000	00:00:00.000	2041-01-06
Incoming SMS	Anne de Marquets	Eudokia Makrembolitissa	Alder Road		10:01:47.000	00:00:00.000	2040-09-21
Incoming SMS	Anne Elliot	Eudokia Makrembolitissa	Adler Street		14:49:41.000	00:00:00.000	2040-10-19
Incoming SMS	Anne Hébert	Eudokia Makrembolitissa	27 S Eglinton E Ramp		23:02:31.000	00:00:00.000	2041-02-01
Incoming SMS	Anne Hébert	Eudokia Makrembolitissa	Alameda Avenue		22:33:46.000	00:00:00.000	2041-01-20

Figure 1: Some rows of the DigForASP dataset. The columns reflect the schema (type, caller, callee, street_a, street_b, time, duration, date).

3.2. Data pre-processing

The dataset can not be used as is to mine sequences with ASP. So, data is pre-processed to lead the dataset to a suitable ASP encoding.

As described in Section 2.2, the problem of sequential pattern mining consists in finding frequent and non-empty sequences s, called sequential patterns, from a database of sequences $\mathcal{D}$. The dataset of interest is the one described in Section 3.1. Obviously, in its original state it cannot be considered as a set of sequences but must undergo an intermediate transformation that leads it to be like the databases described in Section 2.3. In short, each line of the original dataset will be transformed into an ASP fact through the *seq_event* atom.

The procedure for transforming the original dataset into sequences of ASP facts is the following. Each row of the dataset has been transformed into a fact *seq_event(t, p, e)* (Listing 3), where *e* represents the item (in our case the event), *p* defines the position of *e* within the sequence *t* (identified by date). The term *p* is important as it allows you to define the order of events within a sequence. More specifically, *e* is made up of the following features:

- *Type*: type of event ("in_sms", "redirect", "out_call", etc.);
- *Caller*: the name of the caller;
- *Callee*: the name of the callee;
- *Street_a*: the geo-location of the event;
- *Street_b*: the geo-location of the event;
- the *(hour, minute, seconds)* triple: indicates the moment in time when the event occurred;
- *Weekday*: the day of the week (0 = Monday, ..., 6 = Sunday);
- *Duration*: duration, expressed in seconds, of the operation described by *Type*.

Depending on the analyst's needs, it is also possible to transform in sequence only certain days, months or years so as to subsequently carry out a more granular analysis.

```
seq_event((1,9,2040),1,(out_call(simple),eudokia_makrembolitissa,florence_violet_mckenzie
   ,acheson_boulevard,acheson_boulevard,(0,12,9),5,10)).
seq_event((1,9,2040),2,(out_call(simple),eudokia_makrembolitissa,florence_violet_mckenzie
   ,acheson_boulevard,ashcott_street,(0,12,50),5,39)).
seq_event((1,9,2040),3,(in_sms(simple),florence_violet_mckenzie,eudokia_makrembolitissa,
   acheson_boulevard,ashby_place,(1,12,8),5,0)).
.
.
seq_event((2,9,2040),1,(in_sms(simple),annie_dillard,eudokia_makrembolitissa,alder_road,
   none,(9,22,26),6,0)).
seq_event((2,9,2040),2,(out_call(simple),eudokia_makrembolitissa,irena_jordanova,
   alexander_muir_road,adenmore_road,(11,55,29),6,82)).
seq_event((2,9,2040),3,(out_call(simple),eudokia_makrembolitissa,irena_jordanova,
   alder_road,abigail_place,(12,17,57),6,39)).
.
.
```

Listing 3: Some facts representing sequences of events in the dataset.

Notice that, with reference to the first two facts in Listing 3, the event e_1 is prior to e_2 since $(p_{e_1} = 1) < (p_{e_2} = 2)$.

The *seq_event* atoms in Listing 3, in this form, are useless for discovering recurring patterns without first making a more granular choice of which patterns to look for. Additional pre-processing is required to create simpler and easier to analyze sequences. The idea is to create sequences whose identifier refers to a particular day describing what events on that day happened. Two types of sequences have been identified:

Communication sequences The event *e* refers to the *(Caller, Callee)* pair.

Localization sequences The event *e* refers to the *(Street_a, Street_b)* pair.

Listing 4 creates sequences of events in the format shown in Listing 5). The input to this script is sequences like the ones shown in 3. Line 6 allows the creation of sequences via the *seq* predicate. Line 8 contains a rule for calculating the number of sequences, whereas Line 10 generates *len_sequence* facts which denote for each sequence its length, *i.e.*, the number of events. The rule at Line 11 calculates the average length of all sequences, which is the average number of events for each sequence. The rule at Line 13 calculates the sequence with the greatest number of events. Finally, at Lines 15-18, the atoms describing the previously mentioned statistics are shown on an output standard (*e.g.*, terminal, screen) respectively.

```
1  % from
2  % seq_event(Date, Seq_position, (Type_op, Caller, Callee, Street_a, Street_b, Time,
     Weekday, Duration))
3  % to
4  % seq(Date, Seq_position, (Caller, Callee))
5
6  seq(Date, Seq_position, (Caller, Callee)) :- seq_event(Date, Seq_position, (_, Caller,
     Callee, _, _, _, _, _)).
7
```

```
number_of_sequences(N) :- N = #count{D : seq(D, _, _)}.

len_sequence(D, L) :- L = #max{P : seq(D, P, _), seq(D1, _, _), D != D1}, seq(D, _, _).
avg_len_sequences(A) :- S = #sum{L, D : len_sequence(D, L)}, number_of_sequences(N), A =
  S/N.

max_len_sequences(D, N) :- N = #max{L : len_sequence(_, L)}, len_sequence(D, N).

#show number_of_sequences/1.
#show max_len_sequences/2.
#show avg_len_sequences/1.
#show seq/3.
```

Listing 4: Generation of communication sequences with ASP.

```
avg_len_sequences(53).
number_of_sequences(164).
max_len_sequences((1,2,2041),129).
seq((1,9,2040),1,(eudokia_makrembolitissa,florence_violet_mckenzie)).
seq((1,9,2040),2,(eudokia_makrembolitissa,florence_violet_mckenzie)).
seq((1,9,2040),3,(florence_violet_mckenzie,eudokia_makrembolitissa)).
.
.
seq((2,9,2040),1,(annie_dillard,eudokia_makrembolitissa)).
seq((2,9,2040),2,(eudokia_makrembolitissa,irena_jordanova)).
seq((2,9,2040),3,(eudokia_makrembolitissa,irena_jordanova)).
.
.
```

Listing 5: Output generated by Listing 4 from the facts reported in Listing 3.

A similar transformation is needed in order to create localization sequences as shown in Listing 6.

```
avg_len_sequences(53).
number_of_sequences(164).
max_len_sequences((1,2,2041),129).
seq((1,9,2040),1,(acheson_boulevard,acheson_boulevard)).
seq((1,9,2040),2,(acheson_boulevard,ashcott_street)).
seq((1,9,2040),3,(acheson_boulevard,ashby_place)).
.
.
seq((2,9,2040),1,(alder_road,none)).
seq((2,9,2040),2,(alexander_muir_road,adenmore_road)).
seq((2,9,2040),3,(alder_road,abigail_place)).
.
.
```

Listing 6: Output generated by Listing 3.2 from the facts reported in Listing 3.

This can be done by replacing the rule in line 6 of Listing 4 with the following:

```
seq(Date, Seq_position, (Street_a, Street_b)) :- seq_event(Date, Seq_position, (_,_,_,
  Street_a,Street_b,_,_,_)).
```

3.3. Our ASP Encoding for the Analysis of Mobile Phone Records

For the purposes of law enforcement investigations, it is especially useful to understand what the extracted patterns are and what information they provide to the analyst. To this aim, the basic algorithm provided by [10] was modified in such a way as to elaborate patterns whose items have a more complex structure including elements such as caller, callee, type of operation, and time when this occurred (see Section 3.2).

Listing 7 reports the adapted basic algorithm to handle items with an internal structure (Line 1). Consequently, all the rules that managed the embedding were modified to manage a complex item (Lines 11 and 13). For investigative purposes it is necessary to understand in which and how many daily sequences the patterns were found (Lines 21 and 34). Lines 22 and 35, on the other hand, allows you to associate each pattern found with information such as: type of operation (*Type*) carried out between the two communicating entities (*CC*) and the precise time of day (*Time*) with the relative date (*T*). Furthermore, since the dataset contains rows with undefined values (indicated with none), two constraints have been added to eliminate all patterns with a value of none (Lines 25 and 26). A further modification concerns the possibility of being able to search for patterns between a certain minimum and maximum length. To do this, in addition to the *maxlen* parameter, already present, the *minlen* parameter with relative constraint has been added (Line 29).

```
1   item(I) :- seq(_, _,(I, _, _)).
2
3   %sequential pattern generation
4   patpos(1).
5   { patpos(X+1) } :- patpos(X), X<maxlen.
6   patlen(L) :- patpos(L), not patpos(L+1).
7
8   1 {pat(X,I): item(I)} 1 :- patpos(X).
9
10  %pattern embeddings
11  occ(T,1,P) :- seq(T,P,(I, _, _)), pat(1,I).
12  occ(T,L,P) :- occ(T, L,   P-1), seq(T,P,_).
13  occ(T,L,P) :- occ(T, L-1, P-1), seq(T,P,(C, _, _)), pat(L,C).
14
15  %frequency constraint
16  seqlen(T,L) :- seq(T,L,_), not seq(T,L+1,_).
17  supp(T) :- occ(T, L, LS), patlen(L), seqlen(T,LS).
18  :- { supp(T) } < th.
19
20  %pattern information
21  len_support(N) :- N = #count{T : supp(T)}.
```

```
pat_information(T, (Pos, CC) , Type, Time) :- supp(T), pat(Pos, CC), seq(T, P, (CC, Type,
    Time)), occ(T, Pos, P).

% constraint for specific db with none line
:- pat(_, (none, _)).
:- pat(_, (_, none)).

% constraint for pattern of minimum lenght
:- #count{T : pat(T, _)} < minlen.

% atom to print
#show pat/2.
#show len_support/1.
#show support/1.
#show pat_information/4.
```

Listing 7: Modified ASP encoding for sequential pattern mining.

Each answer set returned by the ASP encoding in Listing 7 is a sequential pattern represented by means of the $pat/2$ predicate. The answer includes addition information which is deemed useful for investigation in forensic practice such as: the days in which that pattern was found (see predicate $support/1$), the type of operation carried out between caller and callee and the precise time of the day (see predicate $pat\_information/4$), and the support of that pattern (predicate $len\_support/1$). The support can be useful to understand if the pattern is of interest or a fact given that it occurs every day.

```
Answer: 1
pat(1,(margaret_hasse,karen_cook_mcnally))
pat(2,(karen_cook_mcnally,lucie_julia))
support((8,9,2040)) support((9,9,2040)) support((12,9,2040))
pat_information((8,9,2040),(1,(margaret_hasse,karen_cook_mcnally)),in_sms(simple)
    ,(1,0,55))
pat_information((8,9,2040),(1,(margaret_hasse,karen_cook_mcnally)),in_sms(simple)
    ,(1,2,27))
pat_information((8,9,2040),(2,(karen_cook_mcnally,lucie_julia)),out_sms(simple),(8,55,9))
pat_information((8,9,2040),(2,(karen_cook_mcnally,lucie_julia)),out_sms(simple),(8,55,16)
    )
pat_information((9,9,2040),(1,(margaret_hasse,karen_cook_mcnally)),in_sms(simple)
    ,(1,33,29))
pat_information((9,9,2040),(2,(karen_cook_mcnally,lucie_julia)),out_call(simple)
    ,(10,24,9))
pat_information((12,9,2040),(1,(margaret_hasse,karen_cook_mcnally)),in_call(simple)
    ,(8,23,41))
pat_information((12,9,2040),(2,(karen_cook_mcnally,lucie_julia)),out_call(simple)
    ,(8,26,17))
len_support(3)
```

Listing 8: First of the 15 answers generated by Listing 7.

As an example, the first answer out of the 15 returned by the ASP encoding in Listing 7 is reported in Listing 8. It refers to the running over 100 instances, with maximum pattern length equal to 3 and minimum support threshold equal to 25%. Here, Answer 1 highlights the existence of a sequential pattern which consists of a first communication event between Margaret Hasse and Karen Cook McNally followed by the one between Karen Cook McNally and Lucie Julia (see Lines 2 and 3). The pattern occurs in the days 8, 9 and 12 of September 2040, as shown at Line 4. The fact $len\_support(3)$ provides numerical information about the pattern support, in this case 3. Looking at the facts $pat\_information/4$ concerning the date 08/09/2040 (see Lines 5 and 7), we get to know that Karen (the subject of the phone records) received a text message from Margaret at 01:00:55 and then Karen sent a text message to Lucie at 08:55:09.

4. Experiments

The goal of the following experiments is to evaluate the number of patterns discovered by varying the key parameters. For the first group, the minimum support threshold varies from 10% to 50% while keeping the maximum pattern length fixed at 5. For the second group, the maximum pattern length varies over (3, 5 and 8) while keeping the minimum support threshold fixed to 25%.

All the experiments were conducted over the largest available file of the DigForASP dataset (named Karen Cook McNally) made up of more than 20,000 instances. Given the size, a fairly long execution time for the ASP program is assumed, Therefore, the timeout has been set to 5 hours.

In all presented experiments, we used the version 5.4.0 of clingo, with default solving parameters. The ASP programs were run on a laptop computer with Windows 10 (with Ubuntu 20.04.4 subsystem), AMD Ryzen 5 3500U @ 2.10 GHz, 8GB RAM without using the multi-threading mode of clingo. Multi-threading reduces the mean runtime but introduces variance due to the random allocation of tasks. Such variance is inconvenient for interpreting results with repeated executions.

Table 2
Number of frequent patterns extracted by varying the minimum support threshold (from 10% to 50%) while keeping the maximum pattern length fixed to 5.

Min. Supp. Threshold	# Patterns	Execution time
10%	5,135	18000s (5h)
20%	1,004	18000s (5h)
30%	730	18000s (5h)
40%	55	18000s (5h)
50%	78	18000s (5h)

Table 2 summarizes the results from the first group of experiments. One can observe the decrease in the number of patterns as the minimum support threshold increases. It is interesting to observe the time taken for the computation: in all cases the computation reached 5 hours and stopped. This means that the program did not finish the computation but was interrupted by the set time-out. The reason is to be attributed to the size of the analyzed dataset. Given the nature of ASP (*generate&test* paradigm), the high number of combinations contributed to the long time taken to extract the patterns. Finally the patterns extracted are not all the possible ones but they are only those extracted within 5 hours. There may be others or not by continuing to analyze the search space.

Table 3 summarizes the results from the second group of experiments. Here, it is evident the increase in the number of patterns as the maximum pattern length increases. It is interesting to observe that, as the maximum pattern length increases, the time taken for computation increases as well. Only with a maximum pattern length of 3 the computation ended, whereas for length 5 and 8 the computation was interrupted as soon as the time-out was reached.

Table 3
Number of frequent patterns extracted by varying the maximum pattern length (3, 5, 8) while keeping the minimum support threshold fixed to 25%.

Max. patt. length	# Patterns	Execution time
3	769	4816s (1h20s)
5	922	18000s (5h)
8	1,528	18000s (5h)

4.0.1. Scalability tests

With scalability tests, the goal is to assess the performance of Listing 7 over a dataset of increasing size. Also in this case we considered the dataset concerning Karen Cook McNally, from which we extracted test instances ranging over 100, 1000 and 10,000. We ran two groups of tests, by varying the maximum pattern length from 3 to 5, while keeping the minimum support threshold fixed to 25%. Tables 4 and 5 report the results from the two groups.

Table 4
Number of frequent patterns extracted by varying the number of instances (100, 1K, 10K), while leaving the minimum support threshold (25%) and maximum pattern length (3) unchanged.

# Instances	# Sequences	# Patterns	Execution time
100	6	15	0.079s
1,000	9	9,831	34.962s
10,000	93	947	2468.745s (41m15s)

A similar trend appears evident with a peak of patterns found when the number of instances is equal to 1,000. This peak is due to the fact that the 1,000 instances are distributed over 9 sequences and with a threshold equal to 25% in minimum number of sequences necessary for a pattern to be frequent is equal to 3 (rounded up because ASP considers only integers). As for

Table 5
Number of frequent patterns extracted by varying the number of instances (100, 1K, 10K), while leaving the minimum support threshold (25%) and maximum pattern length (5) unchanged.

# Instances	# Sequences	# Patterns	Execution time
100	6	15	0.101s
1,000	9	127,657	625.498s (10m25s)
10,000	93	2,050	18,000s (5h)

the execution time, we observe that the execution ends before the time-out when the patterns have a maximum length of 3. Conversely, when the maximum pattern length goes from 3 to 5, the execution is interrupted for the time-out set at 5 hours with 10,000 instances while the execution time for 1,000 instances goes from 34 seconds to more than 10 minutes.

5. Final remarks

Sequential pattern mining provides a suite of powerful tools for discovering regularities in sequences of events. Therefore it is naturally suitable for analysing evidence in the context of DF investigations. The expressive power of ASP makes the definition of algorithmic variants of the basic encoding pretty easy, mainly thanks to a clever use of constraints. As a case study we have considered the analysis of a real-world dataset of anonymised phone recordings. The preliminary results are encouraging, although they highlight several weaknesses. A major limit of the current encoding is the combinatorial explosion due to several factors. In order to address this limit, we are currently working on extended versions of the basic ASP encoding here presented, which are aimed at mining so-called condensed patterns (maximal and closed).

For the future we intend to significantly go beyond the state of the art in declarative pattern mining, *e.g.*, by devising effective constraints to reduce the size of the output. In parallel to the methodological work, we plan to solicit a feedback from the DF experts involved in DigForASP, as regards the validity and the usefulness of the work from their viewpoint. The interaction with experts could trigger new interesting directions of research.

Acknowledgments

This article is based upon work from COST Action 17124 "Digital forensics: evidence analysis via intelligent systems and practices (DigForASP)", supported by COST (European Cooperation in Science and Technology). The work is also partially funded by the University of Bari "Aldo Moro" under the 2017-2018 grant "Metodi di Intelligenza Artificiale per l'Informatica Forense".

References

[1] S. Costantini, G. De Gasperis, R. Olivieri, How answer set programming can help in digital forensic investigation, in: D. Ancona, M. Maratea, V. Mascardi (Eds.), Proceedings

of the 30th Italian Conference on Computational Logic, Genova, Italy, July 1-3, 2015, volume 1459 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 53–65. URL: http://ceur-ws.org/Vol-1459/paper29.pdf.

[2] S. Costantini, G. De Gasperis, R. Olivieri, Digital forensics and investigations meet artificial intelligence, Ann. Math. Artif. Intell. 86 (2019) 193–229. URL: https://doi.org/10.1007/s10472-019-09632-y. doi:10.1007/s10472-019-09632-y.

[3] S. Costantini, F. A. Lisi, R. Olivieri, DigForASP: A European cooperation network for logic-based AI in digital forensics, in: A. Casagrande, E. G. Omodeo (Eds.), Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019, volume 2396 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 138–146. URL: http://ceur-ws.org/Vol-2396/paper34.pdf.

[4] J. Han, H. Cheng, D. Xin, X. Yan, Frequent pattern mining: current status and future directions, Data Min. Knowl. Discov. 15 (2007) 55–86. URL: https://doi.org/10.1007/s10618-006-0059-1. doi:10.1007/s10618-006-0059-1.

[5] S. Jabbour, L. Sais, Y. Salhi, Decomposition based sat encodings for itemset mining problems, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2015, pp. 662–674.

[6] T. Guns, A. Dries, S. Nijssen, G. Tack, L. De Raedt, Miningzinc: A declarative framework for constraint-based mining, Artificial Intelligence 244 (2017) 6–29.

[7] B. Negrevergne, T. Guns, Constraint-based sequence mining using constraint programming, in: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2015, pp. 288–305.

[8] M. Gebser, T. Guyet, R. Quiniou, J. Romero, T. Schaub, Knowledge-based sequence mining with asp, in: IJCAI 2016-25th International joint conference on artificial intelligence, AAAI, 2016, p. 8.

[9] L. De Raedt, T. Guns, S. Nijssen, Constraint programming for data mining and machine learning, in: Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010.

[10] T. Guyet, Y. Moinard, R. Quiniou, T. Schaub, Efficiency analysis of asp encodings for sequential pattern mining tasks, in: Advances in Knowledge Discovery and Management, Springer, 2018, pp. 41–81.

[11] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103. URL: http://doi.acm.org/10.1145/2043174.2043195. doi:10.1145/2043174.2043195.

[12] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M. Schneider, Potassco: The potsdam answer set solving collection, Ai Communications 24 (2011) 107–124.

[13] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Clingo= asp+ control: Preliminary report, arXiv preprint arXiv:1405.3694 (2014).

[14] M. Järvisalo, Itemset mining as a challenge application for answer set enumeration, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2011, pp. 304–310.

A Loosely-coupled Neural-symbolic approach to Compliance of Electric Panels

Vito Barbara[1,2], Dimitri Buelli[3], Massimo Guarascio[4], Stefano Ierace[5], Salvatore Iiritano[6], Giovanni Laboccetta[2], Nicola Leone[1], Giuseppe Manco[4], Valerio Pesenti[5], Alessandro Quarta[1,2,7], Francesco Ricca[1] and Ettore Ritacco[4]

[1]*University of Calabria - Via Bucci - 87036 - Rende (CS), Italy*
[2]*DLVSystem s.r.l. - Viale della Resistenza 19/C - 87036 - Rende (CS), Italy*
[3]*Elettrocablaggi s.r.l. - Via Gregorini, 41 - 24065 - Lovere (BG), Italy*
[4]*ICAR-CNR - Via Bucci, 8/9C - 87036 - Rende (CS), Italy*
[5]*Intellimech - Via Stezzano, 87 – 24126 - Bergamo (BG), Italy*
[6]*Revelis s.r.l. - V. J. F. Kennedy, 126 - 87036 - Rende (CS), Italy*
[7]*Sapienza, Università di Roma - Piazzale Aldo Moro 5 - 00185 - Roma (RM), Italy*

Abstract
This paper presents an ongoing work on project MAP4ID "Multipurpose Analytics Platform 4 Industrial Data", where one of the objectives is to propose suitable combinations of machine learning and Answer Set Programming (ASP) to cope with industrial problems. In particular, we focus on a specific use case of the project, where we combine deep learning and ASP to solve a problem of compliance to blueprints of electric panels. The use case data was provided by Elettrocablaggi srl, a SME leader in the market. Our proposed solution couples an object-recognition layer, implemented resorting to deep neural networks, that identifies components in an image of an electric panel, and sends this information to a a logic program, that checks the compliance of the panel in the picture with the blueprint of the circuit.

Keywords
Answer Set Programming, Neural-symbolic AI, Compliance

1. Introduction

With the rise of new technologies for Cyber-Physical Systems and Big Data Analytics, the industry moved a step forward to a new era in the field of manufacturing. This complex transformation, including the integration of emerging paradigms and solutions (e.g., *Machine and Deep Learning*, *Human-Computer Interaction*, *Cloud Computing* and *Industrial Internet Of Things* (IIoT) and *Blockchain*), is referred as *Industry 4.0*. In this evolving scenario, Quality

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
barbara.vito@unical.it (V. Barbara); dimitri.buelli@elettrocablaggi.it (D. Buelli); massimo.guarascio@icar.cnr.it (M. Guarascio); stefano.ierace@intellimech.it (S. Ierace); salvatore.iiritano@revelis.eu (S. Iiritano); laboccetta@dlvsystem.it (G. Laboccetta); nicola.leone@unical.it (N. Leone); giuseppe.manco@icar.cnr.it (G. Manco); valerio.pesenti@intellimech.it (V. Pesenti); alessandro.quarta@uniroma1.it (A. Quarta); ricca@mat.unical.it (F. Ricca); ettore.ritacco@icar.cnr.it (E. Ritacco)
0000-0001-7244-1778 (V. Barbara); 0000-0001-7711-9833 (M. Guarascio); 0000-0002-9742-1252 (N. Leone); 0000-0001-9672-3833 (G. Manco); 0000-0001-6319-2466 (A. Quarta); 0000-0001-8218-3178 (F. Ricca); 0000-0003-3978-9291 (E. Ritacco)

Control (QC) is greatly benefiting from the adoption of advanced Artificial Intelligence (AI) solutions, indeed AI techniques and tools can allow to speed-up or automatize processes of assessment about the integrity, the working capability and the durability of the products. In particular, automating the compliance verification process for products represents an important problem for all the companies operating in the manufacturing field since it is an indispensable but expensive and time consuming task in the supply chain. Recently, defect detection for electrical control panels (ECPs) is gaining growing interest as these tools are used in different scenarios. Indeed, to date ECPs are employed to control a wide variety of components exploited in industry e.g., they permits to control mechanical equipment, electrical devices, etc. In this work, we consider the problem to assist the human operator in verifying the *compliance* of blueprints with the control panel instances so to timely detect possible errors such as missing components, wrong connections and placements, etc. To the best of our knowledge the problem addressed in this paper has been scarcely investigated in the literature. However, some recent works studied tasks relevant for Industy 4.0 within the Predictive Maintenance field. For example, in [1] the authors introduce a framework that integrates Industrial Internet of Things (IIoT) devices, neural networks, and sound analysis for detecting anomalies in the supply chain. [2] proposes a holistic solution for quality inspection based on merging Machine Learning techniques and Edge Cloud Computing technology. A Deep Learning based approach for monitoring the process of sealing and closure of matrix-shaped thermoforming food packages is proposed in [3]. [4] defines a deep neural network (DNN) soft sensor enabling a fast quality control for the Printing Industry. Finally, in [5] the authors describe a deep learning based framework to detect/recognize machines for smart factories. In this paper we devise a novel approach integrating Machine Vision (MV) and Answer Set Programming (ASP) [6] to support the QC for electrical control panels. ASP is a well-established paradigm for declarative programming and non-monotonic reasoning developed in the area of Knowledge Representation and Reasoning [7, 6, 8]. ASP has been employed to develop many academic and industrial applications of AI [9, 10]. ASP is based on logic programming and non-monotonic reasoning, and it allows for flexible declarative modeling of search problems, by means of logic programs (collection of rules), whose intended models (answer sets) encode solutions [11]. In our case, we propose solution composed of two main phases: *(i)* first, we defined a Machine Learning flow based on a neural architecture to address the problem of recognizing the components (*Object Detection*) from the pictures of the panels, *(ii)* then, we realized an Answer Set Programming-based system used to compare the scheme reconstructed from the picture with its original blueprints, to discover possible mismatches/errors. The development of this work was inspired by Elettrocablaggi srl, a SME leader in the market of electric panels. This is one of the use cases of the MAP4ID "Multipurpose Analytics Platform 4 Industrial Data" project that aims at proposing suitable combinations of machine learning and ASP to cope with industrial problems. This paper, after presenting an overview of the architecture of our system for the compliance of electric panels, focuses on the logic-programming-based module of our approach.

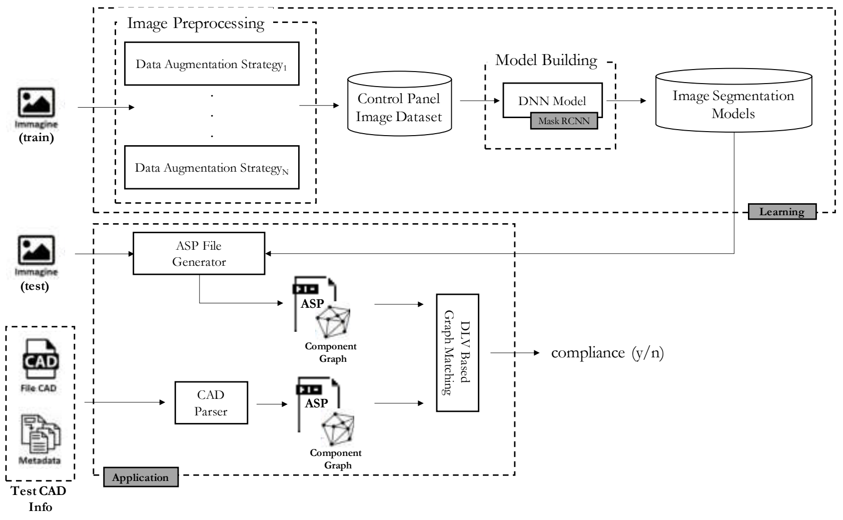


Figure 1: Framework for Automatic Compliance Verification.

2. Framework Overview

In this section we describe the solution approach devised to tackle the main problem i.e., automating the compliance verification process of the control panels. To this aim, we defined the framework shown in Figure 1 that includes two main macro-modules respectively named, *Component Detection* and *Quality Assessment*. The former block is devoted to recognizing the electrical components assembled in the cabinet. Basically, it includes a number of machine learning methods to train a model able to identify the components composing the panel from its picture. The *Model Building* module in Figure 1 allows for training the deep architecture used to perform the component detection. Basically, we used the Convolutional neural architecture proposed in [12], named *Mask R-CNN*, whose objective is to detect and highlight relevant items within an image.

The backbone of the Mask R-CNN used in our framework is a ResNet (Residual Network) [13], whose advantage is the generation of *skip connections* and *residual blocks*, whose usage allows for handling the well-known degradation problem (i.e., neural networks performing worse at the increasing depth), and ensures a good trade-off between convergence rapidity and expressivity/accuracy.

The Quality assessment module exploits ASP to tackle the task of compliance checking. It automatically compares the control panel scheme built starting from the neural network output and the corresponding blueprint to highlight any anomaly. The ASP-based module will be described in details in the following.

3. ASP-based compliance checking

In this section we describe the logic-based component of our architecture for compliance checking. The specification (logic program) described in the following can be fed to an ASP system to actually compute the solutions to the modeled program [14]. In the following we focus on the core parts of our solution and simplify some technical aspects that do not impact on the comprehension of the working principle of our solution. This is done with the aim of making the presentation more accessible and to meet space requirements. Hereafter, we assume the reader to be familiar with ASP, for more details refer to [6, 7, 8].

3.1. Input specification

In ASP the input specification is made by a set of "facts", that is assertions that model true sentences. Thus, the labelled blueprint of the circuit (we informally refer to it as cad) and output of the deep learning algorithm used to recognize the components and the output of the Mask-RCNN (we informally refer to it as net) is converted in a set of ASP facts of the following form:

```
object(LABEL, ID, X_TOP_L, Y_TOP_L, X_BOT_R, Y_BOT_R, MEMBERSHIP).
```

These facts provide information about the components like their label, id, and the top-left and the bottom-right coordinates and the membership. In particular, the membership is valued with "cad" if the object modeled is part of the blueprint of the panel, and "net" if it is recognized by the neural network in the actual picture we are comparing the blueprint.

Moreover, we also compute a graph of topological relations among objects, providing information on relative position and distance among objects. The relative position and the distance among components are actually calculated by our ASP program but for simplicity, we assume here they are given in input as facts of the form:

```
between(ID, START_ID, END_ID, DIR, MEMBERSHIP).
manhattan(ID1, ID2, DIST, MEM1, MEM2).
```

The predicate between denotes the neighbours for the component ID along the direction DIR; while the predicate manhattan specifies the manhattan distance between the two components ID1 and ID2, where the terms MEM1 and MEM2 stand for their membership.

3.2. ASP program

We now present ASP program (see Program 1) that encodes in a uniform way (w.r.t. the input instance provided as set of facts) the compliance problem.

First, the graph is preprocessed (lines 2-3), by calculating useful information about the relative positions of the objects. Next, according to the "guess-and-check" programming methodology a disjunctive rule guesses the mapping between "cad" components of the blueprint and "net" components predicted by the neural network (see lined 6-7). The disjunctive rule can be read as follows: 'Given a cad component and a net component of the same type, the two can be mapped, or not". The candidate solutions are filtered out by the constraints in lines 9-13, ensuring that the same element of the cad is not mapped twice, and the same element of the net is not mapped twice. The optimal mapping is obtained by weak constraints in lines 15-35. In detail, the program first minimizes the cad elements without a mapping (lines 15-16), then (also in order of

Algorithm 1 ASP program modeling compliance

```
 1: % Calculate auxiliary information
 2:       previous(ID, Start_ID, D, M):- between(ID, Start_ID,_ , D, M).
 3:       after(ID, End_ID, D, M):- between(ID, _, End_ID, D, M).
 4: % Guess mapping between cad components and net components
 5:       simpObject(C1,ID1,M) :- object(C1,ID1,_,_,_,_,M).
 6:       mapped(ID1,ID2) ‖ noMapped(ID1,ID2)
 7:                 :- simpObject(C1,ID1,”cad”),simpObject(C1,ID2,”net”).
 8: % No element from the cad is mapped twice
 9:       :- mapped(Cad_ID,Net_ID1), mapped(Cad_ID,Net_ID2),
10:          Net_ID1!=Net_ID2.
11: % No element from the net is mapped twice
12:       :- mapped(Cad_ID1,Net_ID), mapped(Cad_ID2,Net_ID),
13:          Cad_ID1!=Cad_ID2.
14: % Minimize the cad elements without a mapping
15:       atLeastOne(Cad_ID) :- mapped(Cad_ID,_).
16:       :~ simpObject(C1,ID1,”cad”), not atLeastOne(ID1). [1@3,C1,ID1]
17: % Optimize mapping by relative position
18:       :~ mapped(Cad_ID1, Net_ID1), mapped(Cad_ID2,Net_ID2),
19:          previous(Cad_ID1,Cad_ID2,DIR,”cad”),
20:          not previous(Net_ID1, Net_ID2, DIR,”net”).
21:          [1@2,Cad_ID1, Net_ID1,Cad_ID2,Net_ID2,DIR]
22:       :~ mapped(Cad_ID1,Net_ID1), mapped(Cad_ID2,Net_ID2),
23:          after(Cad_ID1, Cad_ID2,DIR,”cad”),
24:          not after(Net_ID1,Net_ID2,DIR,”net”).
25:          [1@2,Cad_ID1,Net_ID1,Cad_ID2,Net_ID2,DIR]
26:       :~ mapped(Cad_ID1, Net_ID1),
27:          previous(Cad_ID1, Cad_ID2, DIR,”cad”),
28:          absent(_,Cad_ID2). [1@2,Cad_ID1,Net_ID1,Cad_ID2,DIR]
29:       :~ mapped(Cad_ID1, Net_ID1),
30:          after(Cad_ID1, Cad_ID2, DIR,”cad”),
31:          absent(_,Cad_ID2). [1@2,Cad_ID1,Net_ID1,Cad_ID2,DIR]
32: % Optimize mapping by distance
33:       :~ mapped(Cad_ID, Net_ID),
34:          manhattan(Cad_ID, Net_ID, Dis,”cad”,”net”).
35:          [Dis@1,Cad_ID,Net_ID,Dis]
36: % Identify absent and in excess components
37:       mappedCad(ID1):- mapped(ID1,_).
38:       mappedNet(ID1):- mapped(_,ID1).
39:       absent(C1,ID1):- simpObject(C1,ID1,”cad”), not mappedCad(ID1).
40:       excess(C1,ID1):- simpObject(C1,ID1,”net”), not mappedNet(ID1).
```

priority) the weak constrains in lines 18-31 ensure that “If a cad component ID1 is mapped to a net component ID2, ID1 neighbors should be mapped to ID2 neighbors”. The mapping is further optimized considering distance (lines 33-35) between cad components and net components. The distance is optimal when the elements are in the same position in “net” and “cad”. Finally, the program identifies components that are absent or in excess w.r.t. the blueprint by rules in lines 37-40. The actual code used in our system implements others features, such as suggestions on where to place the absent elements in the right position inside the panel, and suggestions on where the misplaced components are expected to be moved. These are also obtained by logic rules that are omitted here to simplify the presentation and focus on the core of the solution.

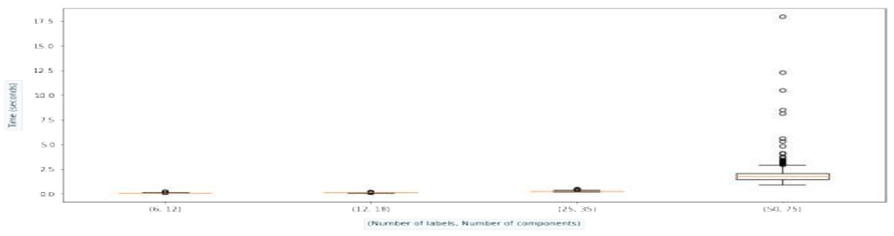


Figure 2: Timing boxplot

3.3. Preliminary results

The component detection module has shown a promising capability in automatically identifying the elements in the electric panels. In an experimentation on a dataset of about 10 thousand pictures (split in training and test sets, with a proportion of 90-10), the Mask R-CNN network was able to detect the 83.1% of all the components. On the other hand, the ASP-based component, that correctly implements our specification, always provides the expected answer, thus accuracy is only determined by the performance of the machine learning component. It might be interesting to know whether the ASP based component is efficient enough. To this end, we conducted an experiment to measure the execution time of the ASP-based component. Usually real panels are made of few components (the larger usually contains less than 25 components). We generated instances of compliance testing in a range of 6 to 50 labels (types of components), and of 12 to 75 components, and averaged over 500 samples the execution time need by DLV2 [15] to solve the instances. The results are reported in Figure 2. It is easy to see that our system can provide optimal answers in a short time, in the order of milliseconds for instances sized as real-world ones, and performance is acceptable (avg. 1.93s, max about 18s) also for instances of 75 components.

4. Conclusion

Our experience confirms that the loose combination of neural networks and ASP can result in an effective solution for checking the compliance of electric panels. The two modules are loosely coupled but complement each-other. Indeed, provided that the ML component knows how to recognize all the components, one can just provide a new logic specification of the blueprint to check compliance, with no need for retraining. As far as future work, we plan to further improve the neural network module to increase its performance, possibly trying to exploit ASP, implement a thorough validation analysis on data provided by Elettrocablaggi, and to develop an online panel compliance system featuring a user interface based on augmented reality glasses.

Acknowledgments

Supported by MISE under project MAP4ID - Multipurpose Analytics Platform 4 Industrial Data” Proj. N. 482 F/190138/01-03/X44.

References

[1] P. Tanuska, L. Spendla, M. Kebisek, R. Duris, M. Stremy, Smart anomaly detection and prediction for assembly process maintenance in compliance with industry 4.0, Sensors 21 (2021) 2376.

[2] J. Schmitt, J. Bönig, T. Borggräfe, G. Beitinger, J. Deuse, Predictive model-based quality inspection using machine learning and edge cloud computing, Advanced Engineering Informatics 45 (2020) 101101. doi:https://doi.org/10.1016/j.aei.2020.101101.

[3] N. Banus Paradell, I. Boada, P. Xiberta, P. Toldra, N. Bustins, Deep learning for the quality control of thermoforming food packages, Scientific Reports 11 (2021) 21887. doi:10.1038/s41598-021-01254-x.

[4] J. Villalba-Diez, D. Schmidt, R. Gevers, J. Ordieres-Meré, M. Buchwitz, W. Wellbrock, Deep learning for industrial computer vision quality control in the printing industry 4.0, Sensors 19 (2019) 3987. doi:10.3390/s19183987.

[5] H. Subakti, J.-R. Jiang, Indoor augmented reality using deep learning for industry 4.0 smart factories, in: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), volume 02, 2018, pp. 63–68. doi:10.1109/COMPSAC.2018.10204.

[6] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[7] C. Baral, Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2003. URL: https://doi.org/10.1017/CBO9780511543357. doi:10.1017/CBO9780511543357.

[8] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–386. doi:10.1007/BF03037169.

[9] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Magazine 37 (2016) 53–68. doi:10.1609/aimag.v37i3.2678.

[10] F. Calimeri, M. Gebser, M. Maratea, F. Ricca, Design and results of the fifth answer set programming competition, Artif. Intell. 231 (2016) 151–181. doi:10.1016/j.artint.2015.09.008.

[11] V. Lifschitz, Answer set planning, in: D. D. Schreye (Ed.), Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999, MIT Press, 1999, pp. 23–37.

[12] K. He, G. Gkioxari, P. Dollár, R. Girshick, Mask r-cnn, in: 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2980–2988. doi:10.1109/ICCV.2017.322.

[13] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[14] Y. Lierler, M. Maratea, F. Ricca, Systems, engineering environments, and competitions, AI Magazine 37 (2016) 45–52. doi:10.1609/aimag.v37i3.2675.

[15] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri, J. Zangari, The ASP system DLV2, in: M. Balduccini, T. Janhunen (Eds.), Logic Programming and Nonmonotonic Reasoning, LPNMR 2017, volume 10377 of *LNCS*, Springer, 2017, pp. 215–221.

KINS: Knowledge Injection via Network Structuring

Matteo Magnini[1,*], Giovanni Ciatto[1] and Andrea Omicini[1]

[1] *Dipartimento di Informatica – Scienza e Ingegneria (DISI)*
Alma Mater Studiorum—Università di Bologna

Abstract

We propose a novel method to inject symbolic knowledge in form of Datalog formulæ into neural networks (NN), called KINS (Knowledge Injection via Network Structuring). The idea behind our method is to extend NN internal structure with ad-hoc layers built out the injected symbolic knowledge. KINS does not constrain NN to any specific architecture, neither requires logic formulæ to be ground. Moreover, it is robust w.r.t. both lack of data and imperfect/incomplete knowledge. Experiments are reported to demonstrate the potential of KINS.

Keywords

neural network, explainable AI, symbolic knowledge injection, KINS, PSyKI

1. Introduction

Supervised machine learning (ML) commonly exploits *opaque* predictors – such as neural networks (NN) – as black boxes [18]. There are several application scenarios where this is becoming troublesome. Indeed, it is non-trivial to forecast what will NN actually learn from data, or whether and how they will grasp general, reusable information for the whole domain. Current state-of-the-art solutions address this issue by supporting a plethora of methods for "opening the black-box" [14]—i.e., inspecting or debugging the inner functioning of NN.

Rather, in this work we tackle the problem of how *injecting* prior *symbolic* knowledge in order to endow them with the designer's common sense. In this way the issue of opacity is circumvented, as designers may force NN to learn correct-by-design information whenever the situation at hand requires to do so.

Along this line, we propose a novel method for the injection of logic formulæ in Datalog [1] form into NN of arbitrary structure. Our method – called KINS (Knowledge Injection via Network Structuring) – works by extending NN architecture with additional *modules*, i.e., ad-hoc layers reflecting symbolic knowledge. The modules are in charge of numerically computing the truth degree of the logic formulæ to be injected, hence increasing the networks performance in the inference phase. Of course, the network still requires training over data in order to adapt injected knowledge to the particular situation at hand.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

*Corresponding author.

matteo.magnini@unibo.it (M. Magnini); giovanni.citatto@unibo.it (G. Ciatto); andrea.omicini@unibo.it (A. Omicini)
https://apice.unibo.it/xwiki/bin/view/MatteoMagnini (M. Magnini); https://about.me/gciatto (G. Ciatto); http://andreaomicini.apice.unibo.it (A. Omicini)
0000-0001-9990-420X (M. Magnini); 0000-0002-1841-8996 (G. Ciatto); 0000-0002-6655-3869 (A. Omicini)

Unlike other knowledge injection techniques, KINS *(i)* does not require input formulæ to be *ground*, *(ii)* does not impose any constraint on the NN, and *(iii)* is robust w.r.t. the lack of data exemplifying the injected knowledge. In other words, KINS supports the injection of knowledge describing scenarios where few training data exist. This in turn may let designers suitably handle the case where poor training data covers a given phenomenon the network should be able to deal with—e.g., unbalanced classes in classification tasks.

In order to validate our method, we report an experiment on a well-known benchmark dataset where the designer's common sense provided by human experts is injected into a NN classifier to improve its performances.

Accordingly, the paper is organised as follows. Section 2 briefly summarises the background on symbolic knowledge injection (SKI). Section 3 formally describes KINS, its rationale and internal operation. Section 4 reports our experiments and their design, whereas results are discussed in Section 5. Finally, Section 6 concludes the paper by providing some insights about how the current limitations of KINS could be overcome.

2. Symbolic Knowledge Injection: Background

We call "symbolic knowledge injection" (SKI) the task of letting a sub-symbolic predictor exploit formal, symbolic information to improve its performances (e.g., accuracy, learning time, need for less training data). Generally speaking, SKI serves the purpose of transferring the designer's common sense into the predictor, hence overcoming the lack of data, or harnessing predictor towards correct-by-construction directions.

While ML predictors are commonly trained over *numeric* data, formal logic enables representation of knowledge in a compact and expressive way, as intensional representations of complex concepts may be concisely written via logic. Hence, assuming that the input–output relation the ML predictor can learn from data can be expressed in formal logic, and that some SKI procedure is available, human experts may handcraft ad-hoc symbolic knowledge to aid the training of a particular predictor, for a specific learning task. In other words, injection makes it possible to provide ML predictors under training with some prior knowledge.

Many methods for SKI have been proposed into the literature along the years [5, 26, 6]. Most of them target NN for their excellent performances in most ML tasks and domains. Concerning the kind of the provided knowledge, it is virtually always expressed in first order logic (FOL) or subsets of FOL such as Horn's clauses, Datalog, knowledge graphs and propositional logic. Possible reasons behind these choices are the flexibility of logic in expressing symbolic information, and the malleability of NN—which can be structured in manifold ways to serve disparate purposes.

Broadly speaking, there exist two major sorts of approaches supporting the injection of symbolic knowledge into NN. Methods of the first sort perform injection during the network's training, using the symbolic knowledge as either a *constraint* or a guide for the optimisation process (i.e., back-propagation). The core idea is to exploit the training step of a NN to increase the error between the prediction value and the expected result when the knowledge is violated. Conversely, approaches of the second sort perform injection by altering the network's architecture to make it mirror the symbolic knowledge.

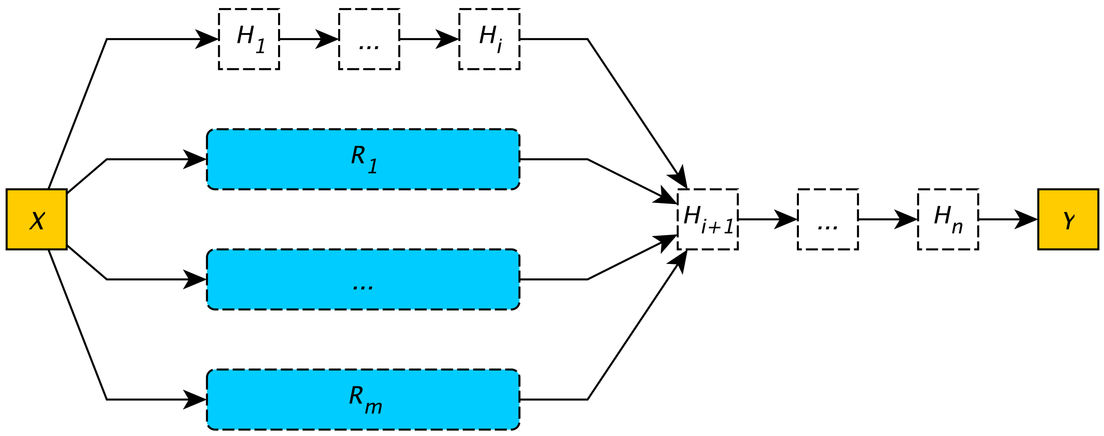


Figure 1: An example of a network's architecture after the insertion of modules derived from logic formulæ.

One of the first notable works that combine NN and logic rules is KBANN [24]. There, given a set of propositional logic rules, a NN is built by mapping each rule into sub parts of the network. In addition, the loss function of the network is modified with a cost factor that penalises the violation of the prior knowledge—so KBANN exploits both the main injection methods. The algorithm is then validated on classification tasks over biological datasets. In Section 4 our SKI algorithm is compared with KBANN by replicating one of those experiments.

Some other interesting works based on NN structuring are [4, 25, 13, 2, 12, 16, 20], whereas relevant works based on constraining are [3, 7, 9, 10, 27]. In particular, the method in [13] is tested on the same task and with the same methodology as [24]. We obtain similar performance (see Section 5), however they report a greater test accuracy value for KBANN and the other benchmark algorithms w.r.t. [24]: we believe that they consider the best result for each algorithm. More details on SKI algorithms can be found in some recent surveys [5, 26, 6].

3. Injection via Network Structuring

We propose an approach to SKI called *KINS*—short for *Knowledge Injection via Network Structuring*. There, a neural network architecture is extended with additional neural *modules*, structured to reflect and mimic the symbolic knowledge provided by designers. There, a module is a (sub-)network having the same input layer of the original network, yet outputting a value representing the evaluation of a logic formula under a continuous interpretation. The model is aimed at *(i)* evaluating a specific logic formula against the current input, and *(ii)* computing the degree of truth of that formula – i.e., a value in range of $[0, 1]$ – to complement the current output. Variables in formulæ are "dynamically grounded" w.r.t. the current input during the feed-forward phase. As a result, non-ground formulæ can be exploited for SKI as such, with no need for any prior groundisation step—which could result unfeasible for non-trivial domains.

It is worth noticing that the provided formulæ are *not* required to cover all possible scenarios. This implies, for instance that rules in classification problems may be provided covering only a portion of all possible classes.

Figure 1 shows the general architecture of the resulting NN after the injection of m modules (represented as blue rectangles), corresponding to the m rules to be injected. Modules can be arbitrarily complex sub-networks, sharing the same input and their final outputs with the original NN. White boxes represent arbitrary hidden layers $H_1, \ldots, H_n$ of the original NN, whereas X is the input layer and Y is the output layer. The injection can be done at any layer H_i and Y. For instance, when dealing with networks that first extract features from the input (such as convolutional NN), then perform classification, one can choose to inject the knowledge in between the two.

Under the hypotheses above, the injection procedure is straightforward. Formulæ are firstly encoded into real-valued functions – hence numerically interpreted –, as described in Section 3.1. Then, a neural module is build to approximate each single real-valued function, following the strategy described in Section 3.2. Finally, that module is added to the original neural network, following the pattern depicted in Figure 1.

Notably, the inner synapses of modules can be either *immutable* – meaning that weights and biases cannot vary during training – or *mutable*—meaning that weights are trainable. Of course, any other synapsis – there including all hidden synapses among layers $H_1, \ldots, H_i$, as well as all the ingoing synapses of layer H_{i+1} and of the following layers – are kept trainable. Thus the NN can exploit both prior knowledge and the information it gathers from data during training. Notice that the synapses connecting each module (and the very last hidden layer) with the output layer are trainable as well. This implies the NN can freely adjust the weights for logic rules during training. The rationale behind this choice is that one cannot assume a logic rule to hold for all the possible patterns in a given domain, yet it may be generally true with a certain degree of confidence. Hence, we let the network learn the relative weight of the injected knowledge w.r.t. the scenario at hand.

In order to operate, KINS does *not* require the loss function to be affected, nor it does impose any constraint on the architecture (e.g., number of layers, number of neurons, types of activation functions, etc.) or the initialisation status (e.g., random weights or partially trained) of the network subject to injection. So, it can be applied to untrained networks as well as to (partially) trained ones. It does require, however, *(i)* the network to have an input and an output layer, and *(ii)* to be trained via gradient descent or similar algorithms. Furthermore, it also requires *(iii)* symbolic knowledge to be expressed via one or more formulæ in Datalog form, and *(iv)* logic statements about the network's input or output features to be encoded.

A public implementation of the algorithm is available as part of the PSyKI framework [19].

3.1. Input Knowledge

KINS supports the injection of knowledge bases composed of one or more logic formulæ in "stratified Datalog with negation" form. Datalog is a restricted subset of first order logic (FOL), representing knowledge via function-free Horn clauses [1]. Horn clauses, in turn, are formulæ of the form $\phi \leftarrow \psi_1 \wedge \psi_2 \wedge \ldots$ denoting a logic implication ($\leftarrow$) stating that ϕ (the head of the clause) is implied by the conjunction among a number of atoms $\psi_1, \psi_2, \ldots$ (the body of the clause). Since KINS relies on Datalog *with negation*, atoms in the bodies of clauses are allowed to be negated. In case the i^{th} atom in the body of some clause is negated, we write $\neg\psi_i$. There, each atom $\phi, \psi_1, \psi_2, \dots$ may be a predicate of arbitrary arity.

An l-ary predicate p denotes a relation among l entities: $p(t_1, \dots, t_l)$ where each t_i is a term, i.e., either a constant (denoted in `monospace`) representing a particular entity, or a logic variable (denoted by *Capitalised Italic*) representing some unknown entity or value. Well-known binary predicates are admissible too, such as $>$, $<$, $=$, etc., which retain their usual semantics from arithmetic. For the sake of readability, we may write these predicates in infix form—hence $> (X, 1) \equiv X > 1$.

Consider for instance the case of a perfect rule (i.e., always true) aimed at defining when a Poker hand can be classified as a pair. Assuming that a Poker hand consists of 5 cards, each one denoted by a couple of variables R_i, S_i – where R_i (resp. S_i) is the rank (resp. seed) of the i^{th} card in the hand –, hands of type *pair* may be described via a set of clauses such as the following one:

$$\begin{array}{rcl} pair(R_1, S_1, \dots, R_5, S_5) & \leftarrow & R_1 = R_2 \\ pair(R_1, S_1, \dots, R_5, S_5) & \leftarrow & R_2 = R_3 \\ & \vdots & \\ pair(R_1, S_1, \dots, R_5, S_5) & \leftarrow & R_4 = R_5 \end{array} \tag{1}$$

To support injection into a particular NN, we further assume that input knowledge base defines at least one outer relation – say *output* or *class* – involving as many variables as the input and output features the NN has been trained upon. The relation may be defined via one clause or more, and each clause may possibly leverage on other predicates in their bodies. In turn, each predicate may be defined through one or more clause. In that case, since we rely on *stratified* Datalog, we require the input knowledge to *not* include any (directly or indirectly) *recursive* clause definition.

For instance, for a 3-class classification task, any provided knowledge base should include a clause, as in the following example:

$$\begin{array}{c} class(\bar{X}, \mathrm{y}_1) \leftarrow p_1(\bar{X}) \land p_2(\bar{X}) \\ class(\bar{X}, \mathrm{y}_2) \leftarrow p_1'(\bar{X}) \land p_2'(\bar{X}) \\ class(\bar{X}, \mathrm{y}_3) \leftarrow p_1''(\bar{X}) \land p_2''(\bar{X}) \end{array}$$

where $\bar{X}$ is a tuple having as many variables as the neurons in the output layer, and y_i is a constant denoting the i^{th} class.

3.2. Fuzzy Logic Formulæ as Neural Modules

Before undergoing injection, each formula corresponding to some output neuron must be converted into a real-valued function aimed at computing the cost of violating that formula. To serve this purpose, we rely on a multi-valued interpretation of logic inspired to Łukasiewicz's logic [15] reported in Table 1.

Accordingly, we encode each formula via $[\![\cdot]\!]$ function, mapping logic formulæ into real-valued functions accepting real vectors of size $m + n$ as input and returning scalars in $\mathbb{R}$ as output.

Formula	C. interpretation	Formula	C. interpretation
$[\![\neg\phi]\!]$	$\eta(1-[\![\phi]\!])$	$[\![\phi \leq \psi]\!]$	$\eta(1+[\![\psi]\!]-[\![\phi]\!])$
$[\![\phi \wedge \psi]\!]$	$\eta(min([\![\phi]\!],[\![\psi]\!]))$	$[\![class(\bar{X}, \mathtt{y}_i) \leftarrow \psi]\!]$	$[\![\psi]\!]^*$
$[\![\phi \vee \psi]\!]$	$\eta(max([\![\phi]\!],[\![\psi]\!]))$	$[\![\mathsf{expr}(\bar{X})]\!]$	$\mathsf{expr}([\![\bar{X}]\!])$
$[\![\phi = \psi]\!]$	$\eta([\![\neg(\phi \neq \psi)]\!])$	$[\![\mathtt{true}]\!]$	1
$[\![\phi \neq \psi]\!]$	$\eta(\lvert[\![\phi]\!]-[\![\psi]\!]\rvert)$	$[\![\mathtt{false}]\!]$	0
$[\![\phi > \psi]\!]$	$\eta(max(0, \frac{1}{2}+[\![\phi]\!]-[\![\psi]\!]))$	$[\![X]\!]$	x
$[\![\phi \geq \psi]\!]$	$\eta(1+[\![\phi]\!]-[\![\psi]\!])$	$[\![\mathtt{k}]\!]$	k
$[\![\phi < \psi]\!]$	$\eta(max(0, \frac{1}{2}+[\![\psi]\!]-[\![\phi]\!]))$	$[\![p(\bar{X})]\!]^{**}$	$[\![\psi_1 \vee \ldots \vee \psi_k]\!]$

* encodes the value for the i^{th} output

** assuming p is defined by k clauses of the form:
$p(\bar{X}) \leftarrow \psi_1,\ \ldots,\ p(\bar{X}) \leftarrow \psi_k$

Table 1
Logic formulæ's encoding into real-valued functions. There, X is a logic variable, while x is the corresponding real-valued variable, whereas is $\bar{X}$ a tuple of logic variables. Similarly, $\mathtt{k}$ is a numeric constant, and k is the corresponding real value, whereas $\mathtt{k}_i$ is the constant denoting the i^{th} class of a classification problem. Finally, $\mathsf{expr}(\bar{X})$ is an arithmetic expression involving the variables in $\bar{X}$.

Scalars are then clipped into the $[0, 1]$ range, via function $\eta : \mathbb{R} \rightarrow [0, 1]$ defined as follows:

$$\eta(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } 0 < x < 1 \\ 1 & \text{if } x \geq 1 \end{cases} \tag{2}$$

The resulting values are the continuous truth degrees of the formulæ. It is worth noticing that this specific mapping is just one among the many that one may design. Therefore, it could be considered a hyperparameter of the algorithm.

While Table 1 describes the mapping between formulæ and their fuzzy interpretations, we discuss how such an interpretation can be further encoded into neural modules to be added to the NN undergoing injection.

By considering the same domain of Equation (1), we can define a perfect rule for class *flush*, i.e., all cards have the same suit, as follow:

$$class(\bar{S}, \mathtt{flush}) \leftarrow S_1 = S_2 \wedge S_1 = S_3 \wedge S_1 = S_4 \wedge S_1 = S_5 \tag{3}$$

The overall procedure that encodes a logic formula into an ad hoc network is exemplified in Figure 3 – where Equation (3) is converted into a neural module –, and it consists of three phases: *(i)* the logic formula is parsed and its abstract syntax tree (AST) is constructed, as shown in Figure 3a; *(ii)* the AST is simplified, merging commutative binary operators, as shown in Figure 3b; and finally *(iii)* the AST is encoded into a NN, where each operator is converted into a neuron reifying the corresponding operation specified in Table 1, as shown in Figure 3c. In particular, in the last step, operators are converted by recursively applying the encoding rules graphically defined in Figure 2. There, variables S_i are mapped into input neurons, while constants possibly occurring in formulæ are mapped into neurons with constant

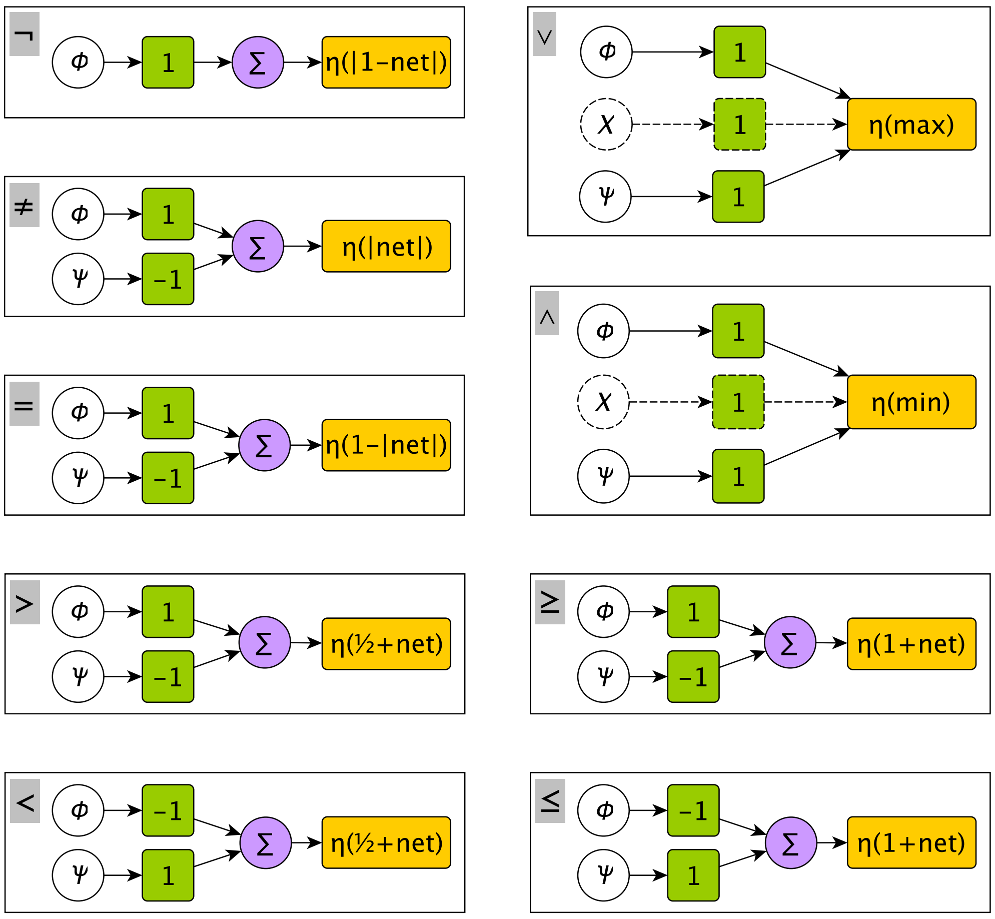
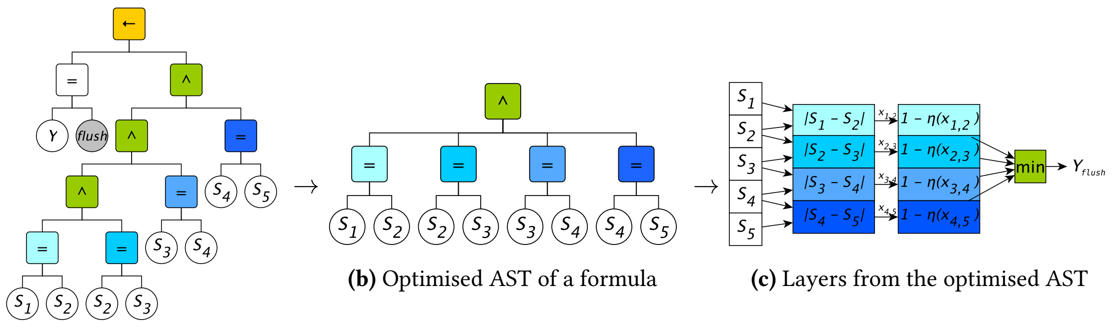


Figure 2: Mapping of formulæ into neurons. White circles are input variables (I), green boxes represent the corresponding weights (W), purple circles are the sum of the weighted inputs ($W \times I$). Yellow rectangles are activation functions, net is the output of $W \times I$, max and min respectively the maximum and minimum of input values, η is the function described in Equation (2).

output. Similarly, algebraic operators such as addition and multiplication are encoded in single neurons that perform the same operation.

4. Experiments

Here we report experiments aimed at assessing KINS for SKI w.r.t. its capability to improve NN's predictive performance. For the sake of reproducibility, the code of our experiments is available at https://github.com/matteomagnini/kins-experiments-cilc-2022.



(a) AST of a formula

(b) Optimised AST of a formula

(c) Layers from the optimised AST

Figure 3: Example of the encoding process of formulæ into module network. Box coloured in the same way represent the encoding of a given operator through each encoding step.

4.1. Primate Splice-Junction Gene Sequences

To validate our method, we test KINS performance on a well-known benchmark: the primate splice-junction gene sequences (PSJGS) dataset [11]. The dataset consists of 3190 records, each of them represents a sequence of 60 DNA nucleotides—namely adenine (`a`), cytosine (`c`), guanine (`g`) and thymine (`t`). Each sequence starts from position -30 up to 30, zero excluded. One DNA sequence can be classified as an intron–exon (`ie`) boundary, an exon–intron (`ei`) boundary, or none (`n`) of them. Class frequencies are 50% for `n`, 25% for both `ie` and `ei`.

The PSJGS dataset comes with a set of textual logical rules aimed at classifying DNA sequences provided by human experts. In Table 2 we report the same rules converted in Datalog form. Datalog rules are equivalent to the original ones that are expressed in a different custom formalism, but they are machine-interpretable as well.

Within Datalog rules, variables are indexed starting from -30 to 30, zero excluded: $X_{-30}, \ldots, X_{-1}, X_{+1}, \ldots, X_{+30}$. There, variable $X_{\pm i}$ denotes the value of the nucleotide in position $\pm i$, which is represented via ad-hoc constants (namely, `a`, `c`, `g`, `t`). For the sake of readability, we write $\bar{X}$ in place of the full sequence of variables $X_{-30}, \ldots, X_{+30}$.

It is worth noticing that the original rules from the PSJGS dataset include different symbols to denote multiple possible nucleotides in a compact way. Table 3 reports the meaning of the additional symbols: rules in Table 2 are reported using them.
When classifying data from the PSJGS dataset according to the rules in Table 2, sequences of type `ie` are correctly classified 295 times – true positives (TP) –, however the rule is also true for 25 `ei` records and for 3 `n` records—false positives (FP). Instead, `ei` sequences are correctly classified 31 times, and there are no FP. Figure 4 shows the confusion matrix of the rules considering also a fictional rule for class `n` that corresponds to the logical *and* of both `ie` and `ei` rules negated:

$$class(\bar{X}, \texttt{n}) \leftarrow \neg class(\bar{X}, \texttt{ei}) \wedge \neg class(\bar{X}, \texttt{ie}) \tag{4}$$

While this is far from being perfect knowledge describing the entire domain with no or few errors, it is still good enough to positively affect the training of the predictor.

Class	Logic Formulation
EI	$class(\bar{X}, \texttt{ei}) \leftarrow X_{-3} = \texttt{m} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g} \wedge X_{+1} = \texttt{g} \wedge$ $X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{a} = \texttt{r} \wedge X_{+4} = \texttt{a} \wedge$ $X_{+5} = \texttt{g} \wedge X_{+6} = \texttt{t} \wedge \neg(ei\_stop(\bar{X}))$
	$ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g}$
	$ei\_stop(\bar{X}) \leftarrow X_{-3} = \texttt{t} \wedge X_{-2} = \texttt{g} \wedge X_{-1} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{a} \wedge X_{-2} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{a} \wedge X_{-2} = \texttt{g}$
	$ei\_stop(\bar{X}) \leftarrow X_{-4} = \texttt{t} \wedge X_{-3} = \texttt{g} \wedge X_{-2} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{a} \wedge X_{-3} = \texttt{a}$
	$ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{a} \wedge X_{-3} = \texttt{g}$
	$ei\_stop(\bar{X}) \leftarrow X_{-5} = \texttt{t} \wedge X_{-4} = \texttt{g} \wedge X_{-3} = \texttt{a}$
IE	$class(\bar{X}, \texttt{ie}) \leftarrow pyramidine\_rich(\bar{X}) \wedge \neg(ie\_stop(\bar{X})) \wedge$ $X_{-3} = \texttt{y} \wedge X_{-2} = \texttt{a} \wedge X_{-1} = \texttt{g} \wedge X_{+1} = \texttt{g}$
	$pyramidine\_rich(\bar{X}) \leftarrow 6 \leq (X_{-15} = \texttt{y} + \ldots + X_{-6} = \texttt{y})$
	$ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{a} \wedge X_{+4} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{a} \wedge X_{+4} = \texttt{g}$
	$ie\_stop(\bar{X}) \leftarrow X_{+2} = \texttt{t} \wedge X_{+3} = \texttt{g} \wedge X_{+4} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{a} \wedge X_{+5} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{a} \wedge X_{+5} = \texttt{g}$
	$ie\_stop(\bar{X}) \leftarrow X_{+3} = \texttt{t} \wedge X_{+4} = \texttt{g} \wedge X_{+5} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{a} \wedge X_{+6} = \texttt{a}$
	$ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{a} \wedge X_{+6} = \texttt{g}$
	$ie\_stop(\bar{X}) \leftarrow X_{+4} = \texttt{t} \wedge X_{+5} = \texttt{g} \wedge X_{+6} = \texttt{a}$

Table 2
Datalog formulæ describing DNA classification criteria generated from the original one.

4.2. Methodology

To make our experiments comparable with already existing literature benchmarks, we follow the very same method used by Towell and Shavlik in [24]. We use 10-fold cross validation with a training size of 1000 randomly-chosen records—drawn among the 3190 available ones (i.e., 31.3% of the overall dataset). Then, for each fold, we train one instance of KINS. Finally, test accuracy is computed on the 2190 records excluded from training, by averaging the predictions of the 10 KINS instances. Unlike the original method, we repeat the overall experiment 30 times – instead of just 10 – to improve result significance.

Symbol	Adenine	Cytosine	Guanine	Thymine	Logic form
`d`	●		●	●	$(X_i = \mathtt{d}) \equiv (X_i = \mathtt{a} \vee X_i = \mathtt{g} \vee X_i = \mathtt{t})$
`m`	●	●			$(X_i = \mathtt{m}) \equiv (X_i = \mathtt{a} \vee X_i = \mathtt{c})$
`r`	●		●		$(X_i = \mathtt{r}) \equiv (X_i = \mathtt{a} \vee X_i = \mathtt{g})$
`s`		●	●		$(X_i = \mathtt{s}) \equiv (X_i = \mathtt{c} \vee X_i = \mathtt{g})$
`y`		●		●	$(X_i = \mathtt{y}) \equiv (X_i = \mathtt{c} \vee X_i = \mathtt{t})$

Table 3
Mapping of aggregative symbols and the four nucleotides. Each symbol can be substituted with one base on the right that has a dot.

More precisely, each time we train an instance of KINS we leverage on a NN with 3 fully connected layers: input layer (60 neurons), hidden layer (neurons), and output layer (3 neurons). During training, we exploit dropout [23] for each layer, up to some extent (0.2), to increase robustness of the network w.r.t. overfitting. Layers have rectified linear unit as activation function, except the output one that has Softmax. The optimiser used for training is Adam [17], categorical cross-entropy as loss function. We use the same stopping criteria used in [24], namely: *(i)* for the 99% of training examples the activation of every output unit is within 0.25 of correct, *(ii)* at most 100 epochs, *(iii)* predictor has at least 90% of accuracy on training examples but has not improved its ability to classify training examples for 5 epochs.

Rules (Table 2) $ei\_stop(\bar{X})$ and $ie\_stop(\bar{X})$ are immutable, while $class(\bar{X}, \mathtt{ei})$, $class(\bar{X}, \mathtt{ie})$ and $pyramidine\_rich(\bar{X})$ are mutable. We recall that mutable rules have trainable weights whereas immutable rules have fixed weights—structure is always preserved.

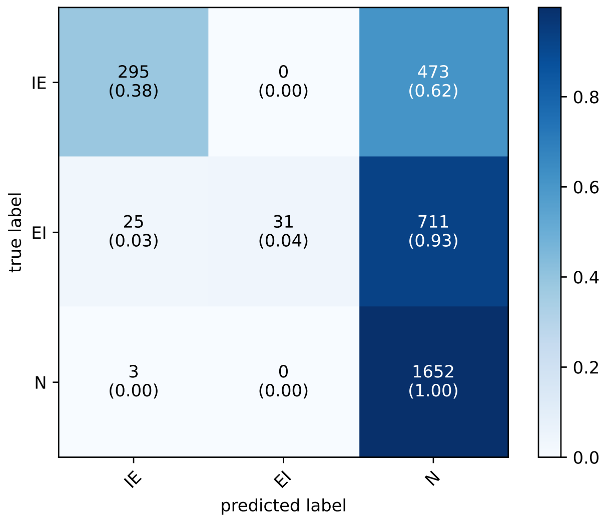


Figure 4: Confusion matrix using only the provided knowledge to classify DNA sequences.

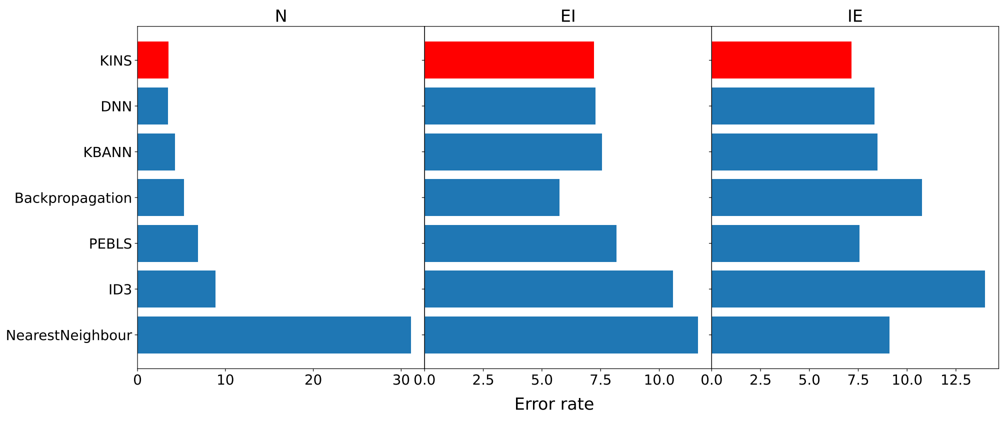


Figure 5: Per-class error rate of different algorithms on the primate slice-junction gene sequences dataset.

5. Discussion

We test KINS by injection the prior knowledge in different layers of the NN previously described. Best results are obtained when the injection is performed at the first hidden layer. Following the aforementioned methodology, we have 10 predictors trained on 900 records each – 1000 unique training records in total out of 3190 – for each experiment. We run 30 experiments using 10-fold cross validation with random weights initialisation.

After the injection of the prior knowledge into the first layer and training, the mean accuracy of the 30 experiment on the test set is 94.73%. Single mean class accuracies are: (`ie`) 92.79%, (`ei`) 92.49%, (`n`) 96.67%.

We execute 30 additional runs using the same base architecture NN *without* the injection of any knowledge obtaining the following results: (mean accuracy) 94.45%, (`ie`) 91.67%, (`ei`) 92.73%, (`n`) 96.54%. After computing Student's T-test on the two distributions we reject the null hypothesis: predictors generated from KINS have better accuracies with statistic relevance.

The improvement of the accuracy using our injection method is significant even with imperfect knowledge. Figure 5 reports the error rate per single class using different algorithms. KINS is our knowledge injection method, while DNN is the network used in KINS, but without knowledge injection. KBANN is the algorithm proposed in [24]: it performs slightly worse than DNN and KINS. Arguably, the main reason for this difference in performance is that the entire structure of KBANN reflects the provided knowledge, whereas in KINS a portion of the network is free to adapt to the data. This is a strength when the knowledge is close to the real rules for the domain, but clearly a weakness in the opposite scenario. The remaining algorithms are *(i)* standard back-propagation [22], *(ii)* PEBLS [8], *(iii)* ID3 [21], and *(iv)* nearest neighbours. Generally, they all perform worse than KINS.

6. Conclusion

In this work we define KINS, a general technique for symbolic knowledge injection into deep neural networks. Designer uses rules in Datalog form (stratified with negation) to express common sense, which are injected through additional modules – ad-hoc layers – capable of evaluating the truth degree of the rules themselves. Rules are interpreted as class-specific fuzzy-logic functions that are then used to build the modules to be inserted into the NN.

We report a number of experiments where we compare networks without knowledge injection with networks – architecturally equivalent except for knowledge injection – that receives additional information in a multi-classification task. We also compare our method with different algorithms, in particular KBANN, which is also based on knowledge injection. The selected task has some of the common criticalities of ML classification tasks, in particular data set size limitation and unbalanced classes. Moreover, the provided prior knowledge is far to be perfect. Results show that our approach can improve network's accuracy with statistical significance.

Investigating the joint use of SKI and symbolic knowledge extraction (SKE) in the same ML workflow is indeed a topic of major interest, which we plan to explore. Introducing multiple cycles of SKI and SKE, possibly using different kind of predictors, could bring several benefits (e.g., final performances of the predictor, more precise knowledge).

Acknowledgments

This paper was partially supported by the CHIST-ERA IV project "Expectation" – CHIST-ERA-19-XAI-005 –, co-funded by EU and the Italian MUR (Ministry for University and Research).

References

[1] Ajtai, M., Gurevich, Y.: Datalog vs first-order logic. Journal of Computer and System Sciences **49**(3), 562–588 (1994). https://doi.org/10.1016/S0022-0000(05)80071-6

[2] Bader, S., Garcez, A.S.d., Hitzler, P.: Computing first-order logic programs by fibring artificial neural networks. In: Russell, I., Markov, Z. (eds.) Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FlAIRS), Clearwater Beach, Florida, USA, May 15–17, 2005. pp. 314–319. AAAI Press (2005), http://www.aaai.org/Library/FLAIRS/2005/flairs05-052.php

[3] Bader, S., Hölldobler, S., Marques, N.C.: Guiding backprop by inserting rules. In: Garcez, A.S.d., Hitzler, P. (eds.) Proceedings of the 4th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy), Patras, Greece, July 21, 2008. CEUR Workshop Proceedings, vol. 366. CEUR-WS.org (2008), http://ceur-ws.org/Vol-366/paper-5.pdf

[4] Ballard, D.H.: Parallel logical inference and energy minimization. In: Kehler, T. (ed.) Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science. pp. 203–209. Morgan Kaufmann (1986), http://www.aaai.org/Library/AAAI/1986/aaai86-033.php

[5] Besold, T.R., d'Avila Garcez, A.S., Bader, S., Bowman, H., Domingos, P.M., Hitzler, P., Kühnberger, K., Lamb, L.C., Lowd, D., Lima, P.M.V., de Penning, L., Pinkas, G., Poon, H.,

Zaverucha, G.: Neural-symbolic learning and reasoning: A survey and interpretation. CoRR **abs/1711.03902** (2017), http://arxiv.org/abs/1711.03902

[6] Calegari, R., Ciatto, G., Omicini, A.: On the integration of symbolic and sub-symbolic techniques for XAI: A survey. Intelligenza Artificiale **14**(1), 7–32 (2020). https://doi.org/10.3233/IA-190036

[7] Che, Z., Kale, D.C., Li, W., Bahadori, M.T., Liu, Y.: Deep computational phenotyping. In: Cao, L., Zhang, C., Joachims, T., Webb, G.I., Margineantu, D.D., Williams, G. (eds.) Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining (KDD), Sydney, NSW, Australia, August 10-13, 2015. pp. 507–516. ACM (2015). https://doi.org/10.1145/2783258.2783365

[8] Cost, S., Salzberg, S.: A weighted nearest neighbor algorithm for learning with symbolic features. Machine learning **10**(1), 57–78 (1993)

[9] Demeester, T., Rocktäschel, T., Riedel, S.: Lifted rule injection for relation embeddings. In: Su, J., Carreras, X., Duh, K. (eds.) Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Austin, Texas, USA, November 1-4, 2016. pp. 1389–1399. The Association for Computational Linguistics (2016). https://doi.org/10.18653/v1/d16-1146

[10] Diligenti, M., Roychowdhury, S., Gori, M.: Integrating prior knowledge into deep learning. In: Chen, X., Luo, B., Luo, F., Palade, V., Wani, M.A. (eds.) Proceedings of the 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, December 18-21, 2017. pp. 920–923. IEEE (2017). https://doi.org/10.1109/ICMLA.2017.00-37

[11] Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml

[12] França, M.V.M., Zaverucha, G., Garcez, A.S.d.: Fast relational learning using bottom clause propositionalization with artificial neural networks. Machine Learning **94**(1), 81–104 (2014). https://doi.org/10.1007/s10994-013-5392-1

[13] d'Avila Garcez, A.S., Zaverucha, G.: The connectionist inductive learning and logic programming system. Appl. Intell. **11**(1), 59–77 (1999). https://doi.org/10.1023/A:1008328630915

[14] Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Computing Surveys **51**(5), 93:1–93:42 (2019). https://doi.org/10.1145/3236009

[15] Hay, L.S.: Axiomatization of the infinite-valued predicate calculus. The Journal of Symbolic Logic **28**(1), 77–86 (1963), http://www.jstor.org/stable/2271339

[16] Hu, Z., Ma, X., Liu, Z., Hovy, E.H., Xing, E.P.: Harnessing deep neural networks with logic rules. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Berlin, Germany, August 7-12, 2016. The Association for Computer Linguistics (2016). https://doi.org/10.18653/v1/p16-1228

[17] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980

[18] Lipton, Z.C.: The mythos of model interpretability. Communications of the ACM **61**(10), 36–43 (2018). https://doi.org/10.1145/3233231

[19] Magnini, M., Ciatto, G., Omicini, A.: On the design of PSyKI: A platform for symbolic knowledge injection into sub-symbolic predictors. In: Calvaresi, D., Najjar, A., Winikoff,

M., Främling, K. (eds.) Explainable and Transparent AI and Multi-Agent Systems – Fourth International Workshop, EXTRAAMAS 2022, Virtual Event, May 9-10, 2021, Revised Selected Papers. Lecture Notes in Computer Science, Springer (2022)

[20] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., De Raedt, L.: Neural probabilistic logic programming in DeepProbLog. Artificial Intelligence **298**, 103504 (2021). https://doi.org/10.1016/j.artint.2021.103504

[21] Quinlan, J.R.: Induction of decision trees. Machine Learning **1**(1), 81–106 (1986). https://doi.org/10.1007/BF00116251

[22] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L.L., PDP Research Group (eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations. Bradford Books (1985), https://mitpress.mit.edu/books/parallel-distributed-processing-volume-1

[23] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014), http://dl.acm.org/citation.cfm?id=2670313

[24] Towell, G.G., Shavlik, J.W.: Knowledge-based artificial neural networks. Artif. Intell. **70**(1-2), 119–165 (1994). https://doi.org/10.1016/0004-3702(94)90105-8

[25] Tresp, V., Hollatz, J., Ahmad, S.: Network structuring and training using rule-based knowledge. In: Hanson, S.J., Cowan, J.D., Giles, C.L. (eds.) Advances in Neural Information Processing Systems 5, NIPS 1992, Denver, Colorado, USA, November 30 – December 3, 1992. pp. 871–878. Morgan Kaufmann (1992), http://papers.nips.cc/paper/638-network-structuring-and-training-using-rule-based-knowledge

[26] Xie, Y., Xu, Z., Meel, K.S., Kankanhalli, M.S., Soh, H.: Embedding symbolic knowledge into deep networks. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada. pp. 4235–4245 (2019), https://proceedings.neurips.cc/paper/2019/hash/7b66b4fd401a271a1c7224027ce111bc-Abstract.html

[27] Xu, J., Zhang, Z., Friedman, T., Liang, Y., Van den Broeck, G.: A semantic loss function for deep learning with symbolic knowledge. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 5498–5507. PMLR (2018), http://proceedings.mlr.press/v80/xu18h.html

Constraint-Procedural Logic Generated Environments for Deep Q-learning Agent training and benchmarking

Stefania Costantini[1], Giovanni De Gasperis[1] and Patrizio Migliarini[1]

[1]*Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, Università degli Studi dell'Aquila*

Abstract

While training and benchmarking neural networks, a large and precise set of data and an efficient test environment are parts of a successful process. A good data set is usually produced with high effort in terms of cost and human work to satisfy the constraints imposed by the expected results. In the first part of this paper we focus on the specification of the properties of the solutions needed to build a data set rather than using common primitives of imperative programming, exploring the possibility to procedurally generate data-sets using constraint programming in Prolog. In this phase geometric predicates describe a virtual environment according to inter-space requirements. The second part is focused to test the generated data set in a machine learning context by means of an AI gym and space search techniques. We developed a deep Q-learning model based neural network agent in Python able to address the NP search problem in the virtual space; the agent has the goal to explore the generated virtual environment to seek for a target, improving its performance through a reinforced learning process.

Keywords

procedural generation, constraint programming, deep Q-learning, intelligent agent, unknown ambient exploration, neural network training, neural network benchmark, neural network gym, AI gym

1. Introduction

The ability to efficiently train and benchmark a deep learning neural network to solve hard tasks such as, speech processing, image recognition, image classification etc. has grown significantly. While those tasks require large - but available - data sets and powerful computing resources, tasks like environment exploration are more difficult to train and benchmark due to lack of available data sets and the long time needed to compile one by hand. To overcome these limitations, we propose a procedural generator of virtual environments based on a logic constraint solver. In the second part of this work, a deep learning explorer agent will be located in the resulting simulated environments to be trained to search for a specific target.

1.1. Related work

Procedural generation is a method of algorithmic data creation widely associated with the world of computer graphics and video games. This context is generally described as Procedural

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

stefania.costantini@univaq.it (S. Costantini); giovanni.degasperis@univaq.it (G. D. Gasperis); patrizio.migliarini@graduate.univaq.it (P. Migliarini)

0000-0002-5686-6124 (S. Costantini); 0000-0001-9521-4711 (G. D. Gasperis); 0000-0002-7824-529X (P. Migliarini)

Generated Content (PGC) *"PCG is the algorithmic creation of game content with limited or indirect user input"* [1]; in this work we focus on the generation of 2D geometric spaces representing a simplified house plan, as shown in Section 2. The procedural generation phase gives us a diverse set of environments which cannot be easily obtained manually. This set can be used to train and benchmark an explorer agent without any given a-priori data set. The PCG is based on search and/or optimization algorithms - i.e. evolutionary algorithms - to find the solution that best satisfies an evaluation metric, which consists of a function that associates each individual component of the generated content with a metric that contributes to the quality of the solution.

1.1.1. Procedural generated spaces

Since we intend to simulate the exploration of an autonomous agent immersed in a virtual space, we focus on procedural generated spaces, i.e., geometrical description of 3D volumes, or 2D partitions that can be assimilated to navigable indoor spaces such as dungeon maps, rooms or house plans. We can find examples of procedural generated spaces in architecture [2]. The dynamic generation video-games dungeons as described in [3] inspired our approach: generate environments that consist of a given number of rooms, connected by a given number of corridors, all enclosed by walls that delimit the possible space.

1.1.2. Constraint generation

Generating content using a constraint problem can be done either using an imperative method [2] where house plans are generated hierarchically over a discrete grid, or using a declarative approach [4] by means of an answer set programming constrained program.

1.1.3. Deep Q-learning for spaces exploration

Deep Q-learning (DQL) is an implementation that substitutes the state-action look-up-table of the classical Q-learning algorithm [5] with a deep learning neural network [6]. It has been recently applied to ambient exploration [7] showing that it is possible to explore unknown environments by an agent that received in input a low resolution image from an on-board simulated camera in the 3D space, while exploring the environment.

2. Geometrical description of 2D environments

Differently from conventional dungeon generation, we opted to avoid internal corridors to reduce the complexity of the generated house plans. So, we introduce a simplification about the inter-rooms connection by having rectangular rooms connected with doors all connected to a single central shared room, as shown in Fig. 1. Also, in order to render a more realistic environment we generated a selection of typical home furnishings and their position inside the rooms, taking into account rules like: a bed shall not stand in the middle of the room, a closet shall not impede doors and windows.

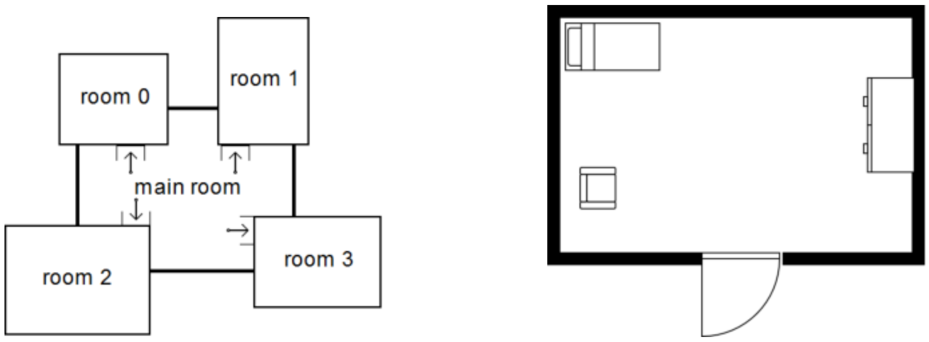


Figure 1: Reference model of the generated rooms set. Left: the room interconnection via the central main room. Right: furniture distribution inside a room of type “bedroom” with a bed, a sofa and a closet.

3. Geometrical Constraints Generation

We defined a set of geometrical constraints that the final virtual house plan has to comply with: room size, room type (bedroom, dining room, bathroom, kitchen), furniture position depending on the room type, allocation of all needed room types to have a complete house.

The set of constraints used to generate the 2D virtual house plans can be summarized as follows:

- the house is made of a central room that connects all the secondary rooms by doors.
- each secondary rooms can have only 1 door.
- all secondary bounding boxes shall not overlap, i.e. have an empty space intersection
- the area of secondary rooms shall never be larger than the main room
- a house shall contain at least 2 rooms of basic types (bedroom, bathroom, kitchen); a shared central room is always generated
- furniture can be positioned only in the secondary rooms
- furniture items have to be compatible with the room type they are in

The rectangles describing the rooms are defined by top-left and bottom-right vertex coordinates in a 2D continuous space.

The problem has been coded in CLP(R) Prolog, *Constraint Logic Programming with Real numbers* [8]; the code is available at the GitHub repository [9], were a full example of the generated Prolog code can be seen at https://github.com/AAAI-DISIM-UnivAQ/bd-procedural-env-deep-learning/blob/master/environments/example1.pl. The constraint generator Python program is called `Main.py` which handles interaction with the user to collect her preferences in terms of number of rooms, type, number and types of furnishings. It then generates the Prolog knowledge base in function of user preferences adding constraints rules; via the PySWIP library it submits the query to the SWI-Prolog interpreter. An example of the generated query is available in the source code repository.

```
generateEnvironment(EnvWidth, EnvHeight, R0X, R0Y, R0W, R0H) :-
   repeat,
```

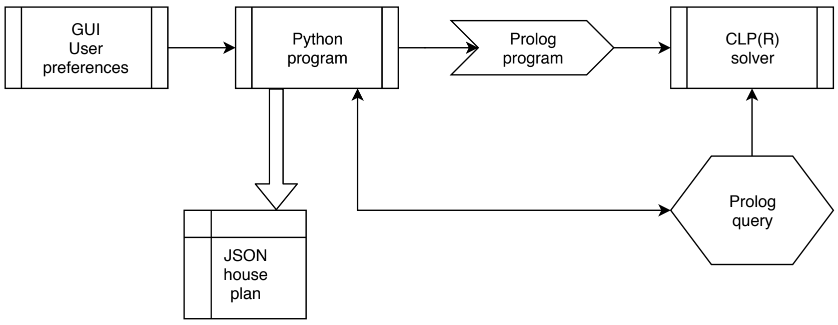


Figure 2: Data flow to obtain the constrained generated virtual house plan: 1: the user fixes the number and type of rooms, with furniture preferences, 2: the Python program generates Prolog rules with CLP(R) constraints expressions, 3: the SWI-Prolog interpreter is invoked and queried, 4: grounded variables from the query result are used to generate the final JSON geometrical house plan description.

```
random(100.0, 145.0, R0W),
random(100.0, 145.0, R0H),
WSUB0 is EnvWidth - R0W,
random(0.0, WSUB0, R0X),
HSUB0 is EnvHeight - R0H,
random(0.0, HSUB0, R0Y),
!.
```

The generated Prolog code above shows the random generation of a room by its origin coordinates (R0X, R0Y) and size (ROW, ROH), respecting the overall space limits (EnvWidth, EnvHeight). It keeps generating random rectangles until an acceptable solution is found according to the following CLP(R) constraints definitions:

```
{(30.0 =< B0X ; B0X + B0W =< 9.5) ;
  (200.0 =< B0Y ; B0Y + B0H =< 175.0)},
{(30.0 =< BS0X ; BS0X + BS0W =< 9.5) ;
  (200.0 =< BS0Y ; BS0Y + BS0H =< 175.0)},
{(30.0 =< W0X ; W0X + W0W =< 9.5) ;
  (200.0 =< W0Y ; W0Y + W0H =< 175.0)},
{(W0X + W0W =< B0X ; B0X + B0W =< W0X) ;
  (W0Y + W0H =< B0Y ; B0Y + B0H =< W0Y)},
{(BS0X + BS0W =< W0X ; W0X + W0W =< BS0X) ;
  (BS0Y + BS0H =< W0Y ; W0Y + W0H =< BS0Y)},
```

This code portion is about the generation of just one bedroom, with bed coordinates starting with B, commode starting with B, closet starting with W. All furniture constraints are in the form described by the formula:

$$\begin{cases} X_{door} + W_{door} \leq X_{obj}; \\ X_{obj} + W_{obj} \leq X_{door}; \\ Y_{door} + H_{door} \leq Y_{obj}; \\ Y_{obj} + H_{obj} \leq Y_{door}; \end{cases} \tag{1}$$

where X and Y are coordinates and W and H are the width and height of the *doors* and *objects* (*objs*) respectively.

The first case represents a situation in which the door of the room is positioned horizontally while, in the second, the door is positioned on a vertical wall. The sub-cases instead represent, in order, the situations in which the object is completely to the right or to the left of the horizontal door or completely above or below the vertical door. It can easily be verified that the simple disjunction of disjunctions ensures the non-overlapping of the elements of a room with its door. A typical graphical result, with much more rooms and constraints, can be seen in Fig. 3 . The colored rendering is the result of a Python/Pygame [1] program that reads the JSON described solution and visualizes it on the computer screen. The temporal complexity of the algorithm is clearly exponential, given that it explores an infinite space of solutions by evaluating the correctness of each one individually. This exploration is based on a random seed, so although the algorithm converges to a solution in a short time for most runs, we cannot currently rule out rare cases where a conforming solution is never found. Furthermore, Prolog queries for the various types of rooms look for a solution that simultaneously satisfies the criteria of each room of that type: this means that, by increasing the number of rooms, the time complexity of the algorithm also increases considerably. In particular, using simple probabilistic terms, called the p probability of finding the solution for a single room, the probability of finding a solution valid for two rooms of the same type is, p^2. More generally this probability is p^{n_i}, with n_i is the numbers rooms for each room type. It is clear that a non-deterministic Turing machine could return any of the valid results in polynomial time as it can attempt every possibility simultaneously, therefore the algorithm can fall into the class of NP-hard problems.

```
{ "roomNumber" : int,
            "floor" : { "x": float, "y": float,
            "width": float, "height": float },
    "RX" : { "x": float, "y": float,
              "width": float, "height": float,
              "type": "bedroom | bathroom | kitchen | hall",
          "children" : [{ "x": float, "y": float,
              "width": float, "height": float,
              "type": "bed | bedside | wardrobe…",
              "orientation": "W | N | E | S" }]
    }
}
```

[1]http://www.pygame.org is Python library optimized in fast screen rendering widely used to implement 2D video games.

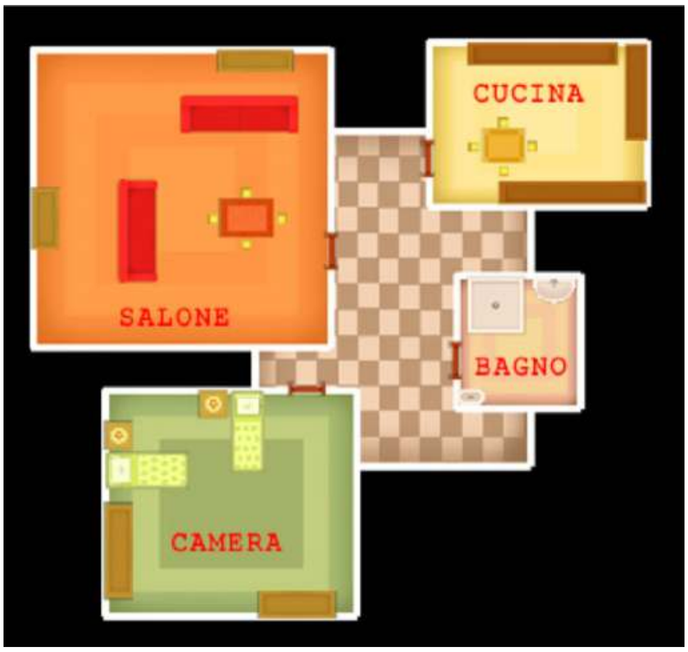


Figure 3: An example of a house plan generated by CLP(R) constraint program. Colored in green is the bedroom, pink is the bathroom, orange is the dining room, yellow is the kitchen.

The above code is the typical JSON definition of a room that is generated by the CLP(R) Prolog system that we called *CoPLEnG* (Costraing-Procedural Logic Environments Generator). Such code is then fed to the simulator were the DQL agent lives in for it to explore.

4. Deep Q-learning explorer agent

The agent starts off in his exploration phase being instantiated by the Python/Pygame program in gym-simulator which takes care of collisions with walls and obstacles and generating sensors data at the agent input layer. The agent has an input array of 40 simulated optical sensors deployed in the front part of its body, in a 120 degree view window; we call them rays, each one measuring the distance to the obstacle/object were is pointed at. Given the 3 possible actions the agent can perform in the environment (rotate left, go straight, rotate right), a totally random selection would have resulted in a 66% chance of rotating against 33% of moving. The observed pattern with these odds was that the agent spent more time wandering around his spawn point rather than exhibiting exploration-oriented behaviours right from the start. So we favored the moving primitive by shifting its chance to as much as 93%, with the rest split among the other 2 actions. This simple change made a huge difference as this modified random agent was now able to move long distances while turning left or right every once in a while, effectively granting a good realism and variability in the forthcoming inputs. In an attempt to raise the neural network from learning unnecessary patterns and thus simplify the model, we have wondered if there were any simpler problems - in fulfilling the main objective - that we could solve in a more mechanical way. Avoiding obstacles and walls was a task which met such description. So,

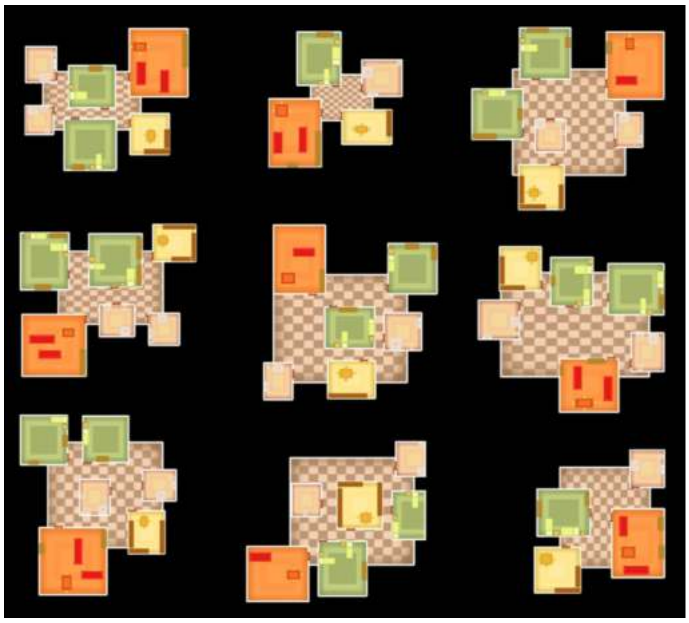


Figure 4: A set of generated home-like spaces derived from the constraints.

drawing inspiration on the Subsumption Architecture introduced by Rodney Brooks [10], we implemented a scripted behaviour - called “Avoidance” - which triggers when the minimum distance perceived in the rays gets lower than an empirically calculated threshold. When it does, it forces the agent to rotate to avoid the incoming collision, purposefully ignoring the prediction from the neural net. With these two mechanics combined, the agent is free to roam with no danger of collision from the beginning and can solidify this initially random behaviour in a predicted one.

4.1. Neural network model

The structure is composed of 7 layers, which follow a so-called “diverging-converging” pattern: the neurons per layer increase in quantity up to half the network and then shrink down to the output layer, whose population is defined by the number of primitive actions, as shown in [11] to have a good compromise between the total number of internal weights, the generalization capability of the neural network and learning and testing performance. The first layer is connected to the input through a pre-processing module; it consists of a vector of 40 tuples bearing the contribution of each ray projected by the agent in the environment. Following is a utility layer (without neurons) called “Flatten” which is particularly useful where the input to the network should consist of a multidimensional vector as it is capable of “spreading” data along a single one-dimensional array to avoid sending through the various layers of “heavy” data to read. The following hidden layers are of the type dense standard and are all activated by the Rectifier Linear Unit (RLU). The function of activation of the output layer is the softmax since the agent shall deliberate over a single motion action . The loss function is the MSE

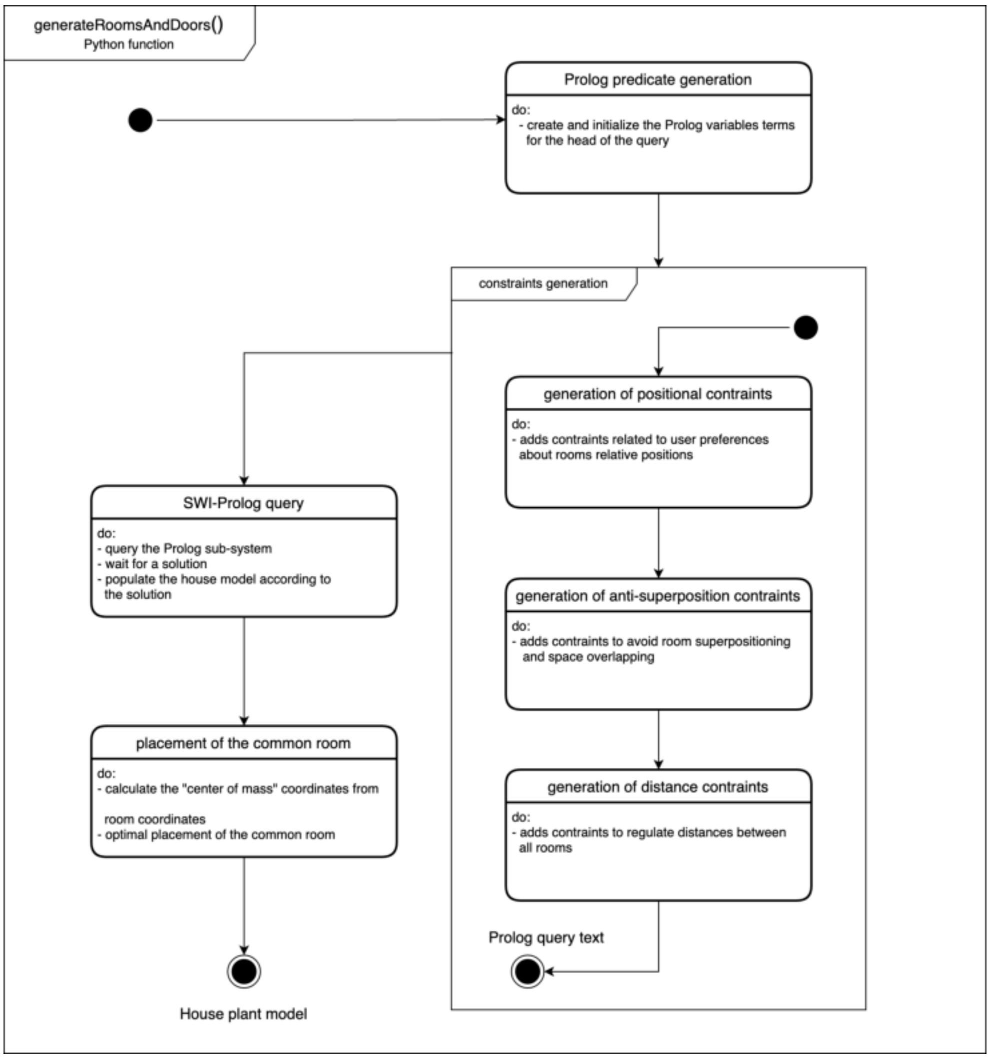

generateRoomsAndDoors()
Python function
Prolog predicate generation
do:
- create and initialize the Prolog variables terms for the head of the query
constraints generation
generation of positional contraints
do:
- adds contraints related to user preferences about rooms relative positions
generation of anti-superposition contraints
do:
- adds contraints to avoid room superpositioning and space overlapping
generation of distance contraints
do:
- adds contraints to regulate distances between all rooms
Prolog query text
SWI-Prolog query
do:
- query the Prolog sub-system
- wait for a solution
- populate the house model according to the solution
placement of the common room
do:
- calculate the "center of mass" coordinates from room coordinates
- optimal placement of the common room
House plant model

Figure 5: The overall flowchart of the generation process performed by the Python function `generateRoomsAndDoors` which, starting from the user preferences describing the expected house plant model, it generates the appropriate query to submit to the SWI-Prolog/CLP(R) interpreter and solver.

(Mean Square Error), in accordance with the formula derived from the Q-Function on which the network optimizes. The deep learning algorithm is driven by the Adam stochastic gradient optimizer [12].

4.2. Performance metric definition

A simple performance metric could be defined by counting the number of targets reached in a finite time interval from its starting position, even if the agent receives commands by a human. We found that, in this kind of generated environments, the performance the human can achieve

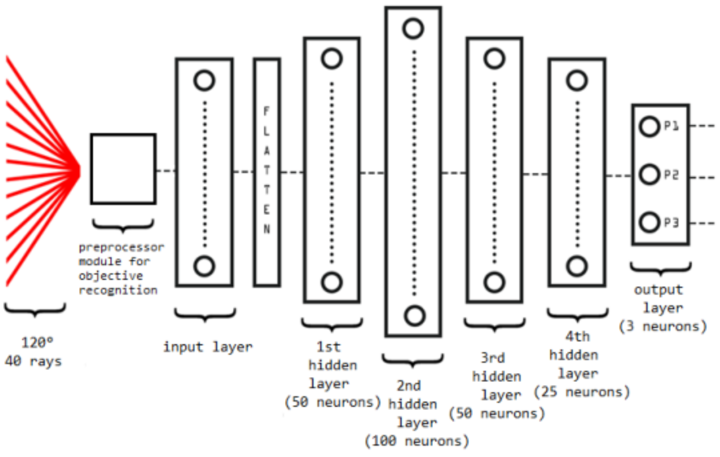

preprocessor module for objective recognition
120° 40 rays
input layer
F L A T T E N
1st hidden layer (50 neurons)
2nd hidden layer (100 neurons)
3rd hidden layer (50 neurons)
4th hidden layer (25 neurons)
P1
P2
P3
output layer (3 neurons)

Figure 6: Deep learning neural network model adopted the controller of the explorer agent. Inputs are proximity distance sensors readings, output is direction of motion or rotation.

is - in average - 27 targets in 15 seconds while searching in the main room, and about 10 targets while searching for them in secondary rooms. In this way we introduced a global behavioural metric to judge the agent performance, not just looking at each single action move.

4.3. Training method

The explorer agent is trained by a reinforcement learning algorithm in function of its performance while exploring the generated environment, as shown in Fig. 7. To further help the model converge, we make use of the well known Remember & Replay method [13]. Experiences are first stored in the agent's memory in the form of tuples containing the information pertaining to a single transition from one observation to the next. The tuple structure is $(state, action, reward, next\_state, done)$ where done is a flag indicating whether the episode has ended or not; this is useful to check if there was any more reward achievable in that time step or not. After an episode ends, a random batch of experiences are sampled from the memory and fed to the neural net for learning. Reward is 1 for each transition in which the agent managed to gather an objective.

5. Results: trained agent performance

The trained agent showed excellent capability to reach the target in the main room, always performing better than the human pilot. On the other, hand statistically it fails to reach the target that is positioned inside the secondary rooms. When the target is visible from the main room it can achieve to reach 1 target in 15 seconds. A trained agent behavior example has been recorded in the animated GIF image inside the software repository [9] .

```
env = loadEnvironment(file_name)
agent = SLAMRobot()
while i = 0 < episodes
   env.reset()
   state = env.projectSegments()
   done = false
   step = 1
   while not done
      action = agent.act(state)
      env.update()
      reward = env.assignReward()
      next_state = env.projectSegments()
      agent.remember(state, action, reward, next_state, done)
      state = next_state
      if step++ == time_steps or env.checkCollision()
         done = true
   agent.replay(batch_size)
```

Figure 7: The pseudo-code for the reinforcement learning of the explorer agent.

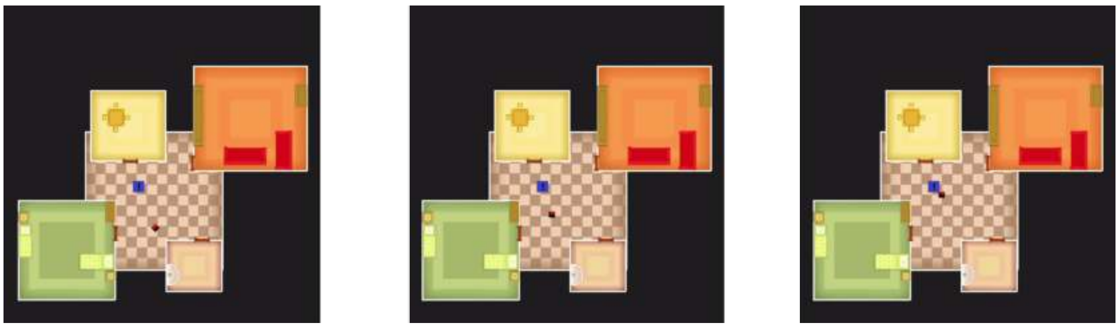


Figure 8: Agent progressing to the objective.

6. Conclusions

In this work we explored the possibility to build a constraint-procedural logic generator for environments to be used as training and benchmarking tool for deep Q-learning agents. The resulting products are a working generator of environments that resembles house floors and an exploring deep Q-learning agent. The generated rooms are connected by a common space and filled with coherent furniture, variably distributed on the room continuous space. The exploring agent is partially capable of solving the given task of finding an object in the generated environment and learning through reinforcement of a reward. It actually outperforms a human competitor when the target is inside the central room. With this work we verified the feasibility of such tool and implemented an instance of both generator and exploring agent. Further evolution of the generator could include support for corridors, multi-story houses and stairs, dynamic elements such as obstacles, simulation for humans, animals and other agents, door states management (such as open, closed, locked etc.), light management, ambience management (such as smoke, fog, humidity etc.), temperature, friction and other challenging elements to train and benchmark the agent. From the agent side, it could be improved with the ability to explore the rooms and interact with the environment (ie. open doors), be able to consider data from other sensors and act accordingly.

7. Acknowledgments

This work started as a joint Bachelor's Degree Thesis in Computer Science @ University of L'Aquila A.Y. 2017-2018 by Leonardo Formichetti and Samuele Petrucci under the supervision of Giovanni De Gasperis. Afterwards, the work has been extended and rationalized by the authors.

References

[1] N. Shaker, J. Togelius, M. J. Nelson, Procedural content generation in games, Springer, 2016.
[2] R. Lopes, T. Tutenel, R. M. Smelik, K. J. De Kraker, R. Bidarra, A constrained growth method for procedural floor plan generation, in: Proc. 11th Int. Conf. Intell. Games Simul, 2010, pp. 13–20.
[3] R. Van Der Linden, R. Lopes, R. Bidarra, Procedural generation of dungeons, IEEE Transactions on Computational Intelligence and AI in Games 6 (2014) 78–89.
[4] A. M. Smith, M. Mateas, Answer set programming for procedural content generation: A design space approach, IEEE Transactions on Computational Intelligence and AI in Games 3 (2011) 187–200.
[5] C. J. Watkins, P. Dayan, Q-learning, Machine learning 8 (1992) 279–292.
[6] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016.
[7] N. Savinov, A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. Lillicrap, S. Gelly, Episodic curiosity through reachability, arXiv preprint arXiv:1810.02274 (2018).
[8] J. Jaffar, S. Michaylov, P. J. Stuckey, R. H. Yap, The clp (r) language and system, ACM Transactions on Programming Languages and Systems (TOPLAS) 14 (1992) 339–395.
[9] G. De Gasperis, P. Migliarini, Environment generator, simulator and neural agent, http://github.com/AAAI-DISIM-UnivAQ/bd-procedural-env-deep-learning, 2019.
[10] R. A. Brooks, Cambrian intelligence: The early history of the new AI, MIT press, 1999.
[11] I. Letteri, G. Della Penna, G. De Gasperis, Botnet detection in software defined networks by deep learning techniques, in: International Symposium on Cyberspace Safety and Security, Springer, 2018, pp. 49–62.
[12] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
[13] L.-J. Lin, Reinforcement learning for robots using neural networks, Technical Report, Carnegie-Mellon University, Pittsburgh, PA, School of Computer Science, 1993.

A Four-State Labelling Semantics for Weighted Argumentation Frameworks[*]

Stefano Bistarelli[1,†], Carlo Taticchi[1,†]

[1]Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Italy

Abstract
Computational Argumentation provides tools for both modelling and reasoning with controversial information. The building blocks in this field are represented by Abstract Argumentation Frameworks, namely structures which explicit the relationships between arguments in order to establish their acceptability. Indeed, arguments can be assigned different justification states: some of the arguments may be accepted, while some other rejected; it could also be the case that some arguments are ignored. Labels corresponding to such states are assigned through sets of criteria called labelling-based semantics. In this paper, we consider Weighted Argumentation Frameworks and propose a novel labelling-based semantics which differentiates four different states, also generalising existing approaches.

Keywords
Computational Argumentation, Weighted Abstract Argumentation Framework, Four-state Labelling

1. Introduction

Computational Argumentation and its applications are receiving increasing interest in many fields of AI. For instance, argumentative processes are used in a paper by Lawrence et al. [1] to interpret online debates, while Walton and Koszowy [2] devise an argumentation system for supporting expert opinion. Argumentation is also used to aid machine learning (as surveyed by Cocarascu and Toni [3]) for both improving performances (e.g., classification accuracy) and providing explanations for the results. Argumentation problems are modelled through Abstract Argumentation Frameworks (AFs in short) [4], which consist of directed graphs in which the nodes are arguments that contain abstract information and the edges represent attack relations. The main goal of these frameworks is to check the acceptability of arguments, which indicates how credible they can be when used, for example, in a speech or debate.

The acceptability of an argument of an AF can be established following different criteria, formalised through the extension-based [4] and the labelling-based semantics [5]. Through the reasoning on the acceptability of the arguments according to a notion of defence, one can divide the set of arguments into two separated subsets, respectively containing acceptable and

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

[*]This work has been partially supported by: GNCS-IndAM, CUP E55F22000270001; Project RACRA - funded by Ricerca di Base 2028-2019, Univeristy of Perugia; Project BLOCKCHAIN4FOODCHAIN - funded by Ricerca di Base 2020, Univeristy of Perugia; Project DopUP - REGIONE UMBRIA PSR 2014-2020.

[†]The author is a member of the INdAM Research group GNCS and of Consorzio CINI.

stefano.bistarelli@unipg.it (S. Bistarelli); carlo.taticchi@unipg.it (C. Taticchi)
0000-0001-7411-9678 (S. Bistarelli); 0000-0003-1260-4672 (C. Taticchi)

non-acceptable arguments. Various approaches have been proposed to cope with the problem of detecting different justification states of arguments in AFs. Indeed, apart from accepted and rejected, arguments could be just ignored or even in an inconsistent state. Caminada [5], for example, introduces a labelling-based semantics in which the state of an argument can be left undecided, without further specifying the reason why. The motivation for not labelling an argument as neither accepted nor rejected is explicitly expressed by Jakobovits and Vermeir [6], who made a distinction between arguments we "don't care" about and those we "do not know" how to label.

In order to increase the expressiveness of AFs, attack relations between arguments can be endowed with a value (a weight) which indicates the strength of the attacks themselves. In this kind of frameworks, called Weighted AF, the acceptability criteria for the arguments also need to consider the weight of incoming and outgoing attacks. Bistarelli et al. [7, 8] group the attacks from an argument to a set of arguments as if they were a unique attack; in particular, the authors consider a weighted notion of defence that takes into account the weight associated with each attack, also generalising other approaches [9, 10].

In this paper, which complements a series of work [11, 12, 13, 14], we provide a four-state labelling for Weighted AFs that generalises other approaches proposed in the literature for the non-weighted case and the three-state labelling for Weighted AFs. For each weighted semantics, we give the conditions under which a labelling corresponds to an extension (that is a set of accepted arguments). We use a partial labelling (i.e., we can leave specific arguments unlabelled) with four labels to identify the possible states of arguments, namely `IN` for accepted, `OUT` for rejected, `DK` for arguments we don't know how to label, and `DC` for arguments we don't care about (because not adopted in an AF or just ignored by the user).

The rest of this paper is structured as follows. In Section 2 we summarise the main concepts of AFs, providing the definitions for extension-based semantics considering both weighted and non-weighted cases. In Section 3 we present our definition of four-state labelling for Weighted Argumentation Frameworks. Section 4 discusses relevant work on labelling-based semantics for (W)AFs already present in the literature, and finally, in Section 5 we conclude the paper, also discussing possible future research lines.

2. Preliminaries

In this section, we recall the formal definitions of AFs [4] and Weighted AFs [7, 8], together with te notion of extension- and labelling-based semantics [15, 5].

Definition 1 (Abstract Argumentation Framework). *Let $\mathcal{U}$ be the set of all available arguments[1]. An Abstract Argumentation Framework is a pair $\langle \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{A} \subseteq \mathcal{U}$ is a set of arguments and $\mathcal{R}$ is a binary relation on $\mathcal{A}$. Arguments in $\mathcal{A}$ are said to be adopted.*

Definition 2 (Attacks). *Let $\langle \mathcal{A}, \mathcal{R} \rangle$ be an AF, and consider two arguments $a, b \in \mathcal{A}$. If $(a, b) \in \mathcal{R}$, we say that a attacks b; conversely, b is an attacker of a. Moreover, given $A \subseteq \mathcal{A}$, we define*

[1]The set $\mathcal{U}$, which we refer to as the *Universe* of arguments, is not present in the original definition of AFs, and it is introduced to model arguments which are external to $\mathcal{A}$ [16, 17].

the sets $a^+ = \{b \in \mathcal{A} \mid (a, b) \in \mathcal{R}\}$, $a^- = \{b \in \mathcal{A} \mid (b, a) \in \mathcal{R}\}$, $A^+ = \bigcup\{a^+ \mid a \in A\}$ *and* $A^- = \bigcup\{a^- \mid a \in A\}$.

In order for an argument a to be acceptable, we require that every attacker of a is defeated in turn by some other argument.

Definition 3 (Acceptable argument). *Let* $\langle \mathcal{A}, \mathcal{R} \rangle$ *be an AF, and consider* $a \in \mathcal{A}$ *and* $D \subseteq \mathcal{A}$. *The argument* a *is acceptable with respect to the subset* D *if and only if* $\forall b \in A . \exists d \in D \mid (b \in a^-) \implies (d \in b^-)$. *In that case, we say that* a *is* ***defended*** *by* D *from the attack of* b.

We also say that argument is acceptable if there exists a subset of arguments with respect to which it is acceptable. Using the notion of defence as a criterion for distinguishing acceptable arguments in the framework, one can further refine the set of selected arguments through the so-called extension-based semantics.

Definition 4 (Extension-based semantics). *Given an AF* $\langle \mathcal{A}, \mathcal{R} \rangle$, *we say that a set of arguments* $E \subseteq \mathcal{A}$ *is conflict-free if and only if* $\nexists a, b \in E$ *such that* $(a, b) \in \mathcal{R}$. *A conflict-free set* E *is said to be*

- *admissible, if each* $a \in E$ *is defended by* E
- *complete, if it is admissible and* $\forall a \in \mathcal{A}$ *defended by* E, $a \in E$
- *stable, if* $E \cup E^+ = \mathcal{A}$
- *preferred, if it is complete and it is maximal (with respect to set inclusion)*
- *grounded, if it is complete and it is minimal (with respect to set inclusion)*

In this paper, we only consider the above semantics, although other extension-based semantics have also been defined in the literature, such as ideal, semi-stable and stage [15]. In Figure 1, we provide an example of an AF for which we compute the set S of conflict-free, admissible, complete, stable, preferred and grounded extensions (abbreviated with cf, adm, com, stb, prf and gde, respectively): $S_{cf}(F) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{b, d\}\}$, $S_{adm}(F) = \{\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$, $S_{com}(F) = \{\{a\}, \{a, c\}, \{a, d\}\}$, $S_{prf}(F) = \{\{a, c\}, \{a, d\}\}$, $S_{stb}(F) = \{\{a, d\}\}$ and $S_{gde}(F) = \{\{a\}\}$.

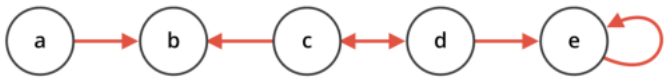


Figure 1: Example of an AF with five arguments.

We give some details on the extensions found. The singleton $\{e\}$ is not conflict-free because e attacks itself. Argument b is not contained in any admissible extension because no other argument (included itself) defends b from the attack of a. The empty set and the singletons $\{c\}$ and $\{d\}$ are not complete extensions because they do not contain a, which is not attacked by any other argument. Only the maximal complete extensions $\{a, c\}$ and $\{a, d\}$ are preferred, while the minimal complete $\{a\}$ is the unique grounded extension. Since argument a attacks arguments b and argument d attacks arguments c and e, we have that $\{a, d\}$ is a stable extension.

To obtain different nuances for the acceptability of arguments, we can rely on the notion of labelling-based semantics [5], namely functions that partitions the arguments of an AF into three subsets.

Definition 5 (Labelling for AFs). *Let $F = \langle \mathcal{A}, \mathcal{R} \rangle$ be an AF. A labelling L of F is a total function $L : \mathcal{A} \rightarrow$ {IN, OUT, UNDEC}.*

Notation 1. *Given a labelling L of $F = \langle \mathcal{A}, \mathcal{R} \rangle$ and $A \subseteq \mathcal{A}$, we denote $A \downarrow_{IN}$, $A \downarrow_{OUT}$ and $A \downarrow_{UNDEC}$ the sets of all arguments labelled IN, OUT and UNDEC, respectively, by L.*

We show in Figure 2 an example of labelling: IN arguments are highlighted in green and OUT ones in red, while UNDEC are represented in yellow. It is also possible to identify a correspondence between labellings and sets of extensions for a certain semantics [15].

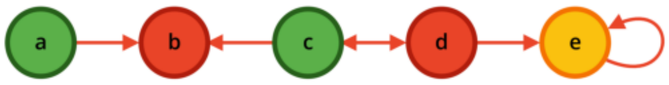


Figure 2: Example of labelling for an AF with five arguments.

Definition 6 (Labelling-based semantics). *Let L be a labelling of an AF $F = \langle \mathcal{A}, \mathcal{R} \rangle$ and $a \in \mathcal{A}$. Then*

- *L is a conflict-free labelling if:*
  - $L(a) = IN \implies a^- \downarrow_{IN} = \emptyset$, *and*
  - $L(a) = OUT \implies a^- \downarrow_{IN} \neq \emptyset$
- *L is a admissible labelling if:*
  - $L(a) = IN \implies a^- = a^- \downarrow_{OUT}$, *and*
  - $L(a) = OUT \implies a^- \downarrow_{IN} \neq \emptyset$
- *L is a complete labelling if:*
  - $L(a) = IN \iff a^- = a^- \downarrow_{OUT}$, *and*
  - $L(a) = OUT \iff a^- \downarrow_{IN} \neq \emptyset$
- *L is a stable labelling if:*
  - *L is a complete labelling, and*
  - $\mathcal{A} \downarrow_{UNDEC} = \emptyset$;
- *L is a preferred labelling if:*
  - *L is an admissible labelling, and*
  - *$\mathcal{A} \downarrow_{IN}$ is maximal among all the admissible labellings*

- *L is a grounded labelling if:*
  - *L is a complete labelling, and*
  - *$\mathcal{A}\downarrow_{IN}$ is minimal among all the complete labellings*

We have, for instance, that the labelling of Figure 2 is complete, but not grounded. Since all attacks in AFs have the same "strength", it is not possible to further diversify the relations among arguments, and thus the existence of an attack is the only thing that matters in determining the semantics. To overcome this limitation, we can resort to Weighted AFs, whose attacks are endowed with a value that represents the support of the relation [18]. In this kind of framework, the notion of defence needs to be adapted to encompass the refined attack relation. In a paper by Bistarelli et al. [8], Weighted AFs are equipped with a c-semiring [19, 20] that provides operations for composing the weights and estimating the effectiveness of a defence.

Definition 7 (c-semirings). *A c-semiring is a tuple $\mathbb{S} = \langle S, \oplus, \otimes, \bot, \top\rangle$ such that S is a set, $\top, \bot \in S$, and $\oplus, \otimes : S \times S \to S$ are binary operators making the triples $\langle S, \oplus, \bot\rangle$ and $\langle S, \otimes, \top\rangle$ commutative monoids (semi-groups with identity), satisfying* i) $\forall s, t, u \in S.\ s \otimes (t \oplus u) = (s \otimes t) \oplus (s \otimes u)$ *(distributivity), and* ii) $\forall s \in S.\ s \otimes \bot = \bot$ *(annihilator). Moreover, we have that $\forall s, t \in S.\ s \oplus (s \otimes t) = s$ (absorption). The operator $\oplus$ also defines a preference relation $\leq_{\mathbb{S}}$ over the set S, such that $a \leq_{\mathbb{S}} b$ if and only if $a \oplus b = b$, for all $a, b \in S$.*

We list some of the most common instances of c-semirings:

- $\mathbb{S}_{boolean} = \langle\{false, true\}, \vee, \wedge, false, true\rangle$
- $\mathbb{S}_{fuzzy} = \langle[0, 1], \max, \min, 0, 1\rangle$
- $\mathbb{S}_{probabilistic} = \langle[0, 1], \max, \times, 0, 1\rangle$
- $\mathbb{S}_{weighted} = \langle\mathbb{R}^+ \cup \{+\infty\}, min, +, +\infty, 0\rangle$

The interval $[0, 1]$ used for $\mathbb{S}_{fuzzy}$ and $\mathbb{S}_{probabilistic}$ is to be considered valid for both real and rational numbers. We denote with WAF$_{\mathbb{S}}$ a Weighted AF endowed with a c-semirings $\mathbb{S}$ and we call it a semiring-based Weighted AF.

Definition 8 (Semiring-based Weighted AF). *Let $\mathcal{U}$ be the set of all available arguments. A semiring-based Weighted AF is a quadruple $\langle\mathcal{A}, \mathcal{R}, W, \mathbb{S}\rangle$, where $\mathcal{A} \subseteq \mathcal{U}$ is the set of adopted arguments, $\mathcal{R}$ the attack relation on $\mathcal{A}$, $W : \mathcal{A} \times \mathcal{A} \to S$ a binary function, and $\mathbb{S}$ a c-semiring $\langle S, \oplus, \otimes, \bot, \top\rangle$.*

The binary function W assigns a weight to attacks between arguments: we use $W(a, b) = s$ to indicate that the attack from a towards b has weight $s \in S$. In our setting, the $\top$ element of a c-semiring (*e.g.*, 0 for the weighted and $true$ for the boolean) denotes the absence of a pair in the relation R. Hence, $(a, b) \in \mathcal{R}$ if and only if $W(a, b) <_{\mathbb{S}} \top$.

Given a WAF$_{\mathbb{S}}$, we can evaluate the overall weight of all the attacks from a set of arguments towards another set through the binary **composition** operator $\otimes$ of the c-semiring $\mathbb{S}$ [7, 21]. In particular, we use $\bigotimes$ to indicate the $\otimes$ operator on a set of values.

Definition 9 (Weighted attacks). *Let $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ be a WAF$_\mathbb{S}$ and consider two sets of arguments $B, D \in \mathcal{A}$. We say that B attacks D, and the weight of such attack is $k \in S$, if*

$$W(B, D) = \bigotimes_{b \in B, d \in D} W(b, d) = k.$$

Following Definition 9, it is also possible to compose the attacks both from a set of arguments towards a single argument and from a single argument towards a set of arguments. We can now express the notion of weighted defence.

Definition 10 (Weighted defence). *Let $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ be a WAF$_\mathbb{S}$. We say that $B \subseteq \mathcal{A}$ w-defends $b \in \mathcal{A}$ if and only if $\forall a \in \mathcal{A}$ such that $(a, b) \in \mathcal{R}$, $W(a, B \cup \{b\}) \geq_\mathbb{S} W(B, a)$.*

Consider the WAF$_\mathbb{S}$ of Figure 3. To verify whether the set $\{a\}$ w-defends d we need to check if $W(c, \{a, d\}) \geq_\mathbb{S} W(\{a\}, c)$. We have that $W(c, \{a, d\}) = 3$ and $W(\{a\}, c) = 2$, and since $3 \not\geq_\mathbb{S} 2$, we conclude that a alone is not sufficient to w-defend d in this example[2]. If we consider the set $\{a, b\}$, instead, we can see that $W(c, \{a, b, d\}) \geq_\mathbb{S} W(\{a, b\}, c)$ since $W(c, \{a, b, d\}) = W(\{a, b\}, c) = 3$, and therefore $\{a, b\}$ w-defends d.

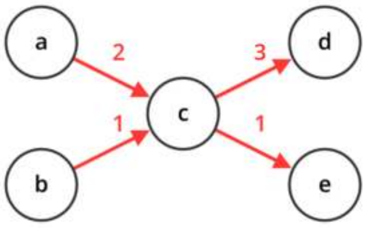


Figure 3: Example of a WAF$_\mathbb{S}$ with $\mathbb{S} = \mathbb{S}_{weighted}$.

Notation 2. *Let $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ be a WAF$_\mathbb{S}$ and consider an argument $a \in \mathcal{A}$. We denote the weight of a set of attacks towards a with $w_{a^- \downarrow_{IN}} = W(a^- \downarrow_{IN}, a)$, and the weight of outgoing attacks with $w_{a^+ \downarrow_{IN}} = W(a, a^+ \downarrow_{IN})$.*

It is then possible to redefine all the extension-based semantics of Definition 4 by using the notion of weighted defence for checking the acceptability of arguments [8].

Definition 11 (Extension-based semantics for WAF$_\mathbb{S}$). *Consider a WAF$_\mathbb{S}$ $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ and a subset of arguments $E \subseteq \mathcal{A}$. We have that E is w-conflict-free if $W(E, E) = \top$. A w-conflict-free subset E is*

- *w-admissible, if $\forall a \in E^-$. $W(a, E) \geq_\mathbb{S} W(E, a)$*
- *w-complete, if it is w-admissible and each $b \in \mathcal{A}$ such that $E \cup \{b\}$ is w-admissible belongs to E*
- *w-stable, if it is w-admissible and $\forall a \notin E$. $\exists b \in E$ such that $W(b, a) <_\mathbb{S} \top$*

[2]We remark that $3 <_\mathbb{S} 2$ when $\mathbb{S} = \mathbb{S}_{weighted}$, i.e., greater means worse.

- *w-preferred, if it is a maximal (with respect to set inclusion) w-admissible subset of $\mathcal{A}$*
- *w-grounded, if it is the maximal (with respect to set inclusion) w-admissible extension included in the intersection of w-complete extensions*

As for the non-weighted case, also sets of acceptable arguments in a WAF$_{\mathbb{S}}$ can be identified through special labelling functions. In the next section, we expand the discussion in this direction, introducing a weighted labelling that differentiates up to four states of acceptability.

3. From Three-State to Four-State Weighted Labelling

The labelling for AFs of Definition 5 and the derived labelling-based semantics are a useful tool which identifies up to three degrees of acceptability for the arguments while maintaining a direct connection with set of extensions for the classical semantics introduced by Dung [4]. However, the labelling function shown in the previous section forces all arguments that are neither IN nor OUT to be labelled UNDEC, thus not allowing to distinguish arguments we don't know how to label from arguments we deliberately decide to ignore. In other words, three labels are not sufficient to express the difference between the possible causes for which an argument can be labelled UNDEC. Consider for instance the AF in Figure 4, whose arguments are labelled according to the admissible labelling-based semantics. Arguments c and d are both labelled UNDEC, but for two distinct reasons: c, which could potentially be accepted (it has no IN attackers), is ignored, while d is attacking itself and thus it can neither be accepted nor rejected. To overcome these inconvenience, more informative labellings have been proposed [22, 6, 23] that split the UNDEC label into two distinct labels, resulting in a total of four recognisable acceptability states[3].

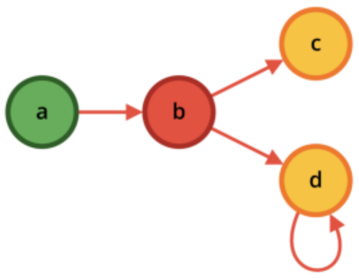


Figure 4: Example of labelling with two UNDEC arguments.

Before introducing our proposal for a labelling function able to work with WAF$_{\mathbb{S}}$ and which makes use of four labels, we recall the definition of three-state weighted labelling [11, 12, 14]. In order to incorporate the notion of weighted defence into the labelling, also the strength of the attack relations is taken into account.

Definition 12 (Three-state Labelling for WAF$_{\mathbb{S}}$). *Let $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ be a WAF$_{\mathbb{S}}$. A three-state labelling L of F is a total function $L : \mathcal{A} \rightarrow$ {IN, OUT, UNDEC}.*

[3] More nuances of acceptability can be enabled through ranking-based semantics [24], however, losing the correspondence with accepted arguments identified by extension-based semantics.

Definition 13 (Three-state labelling-based semantics for WAF$_{\mathbb{S}}$). *Consider a three-state labelling L of $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ and an argument $a \in \mathcal{A}$.*

- *L is a w-conflict-free labelling when*
  - $L(a) = \mathit{IN} \implies a^- \downarrow_{\mathit{IN}} = \emptyset$ *and*
  - $L(a) = \mathit{OUT} \implies a^- \downarrow_{\mathit{IN}} \neq \emptyset$
- *L is a w-admissible labelling for F if and only if:*
  - $L(a) = \mathit{IN} \implies a^- = a^- \downarrow_{\mathit{OUT}} \wedge \forall b \in a^-.\, w_{b^- \downarrow_{\mathit{IN}}} \leq_{\mathbb{S}} w_{b^+ \downarrow_{\mathit{IN}}}$
  - $L(a) = \mathit{OUT} \implies w_{a^- \downarrow_{\mathit{IN}}} <_{\mathbb{S}} \top$
- *L is a w-complete labelling for F if and only if:*
  - $L(a) = \mathit{IN} \iff a^- = a^- \downarrow_{\mathit{OUT}} \wedge \forall b \in a^-.\, w_{b^- \downarrow_{\mathit{IN}}} \leq_{\mathbb{S}} w_{b^+ \downarrow_{\mathit{IN}}}$
  - $L(a) = \mathit{OUT} \iff w_{a^- \downarrow_{\mathit{IN}}} <_{\mathbb{S}} \top$
- *L is a w-stable labelling for F if and only if*
  - *L is a w-complete labelling and*
  - $\mathcal{A} \downarrow_{\mathit{UNDEC}} = \emptyset$
- *L is a w-preferred labelling for F if and only if*
  - *L is a w-admissible labelling and*
  - *$\mathcal{A} \downarrow_{\mathit{IN}}$ is maximal among all the w-admissible labellings*
- *L is a w-grounded labelling for F if and only if:*
  - $L(a) = \mathit{IN} \iff$ *for all w-complete labellings L', $L'(a) = \mathit{IN}$ and*
  - $L(a) = \mathit{OUT} \iff w_{a^- \downarrow_{\mathit{IN}}} <_{\mathbb{S}} \top$

The sets of arguments labelled IN by the labelling-based semantics of Definition 13 are equivalent to extensions of the corresponding semantics. OUT and UNDEC arguments, instead, are considered to be rejected. Our proposal for a richer labelling function is based on four labels, namely IN, OUT, DK and DC.

Definition 14 (Four-State Labelling for WAF$_{\mathbb{S}}$). *Let $\mathcal{U}$ be a universe of arguments and $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ a WAF$_{\mathbb{S}}$ with $\mathcal{A} \subseteq \mathcal{U}$. A four-state labelling L of F is a partial function $L : \mathcal{U} \rightharpoonup \{\mathit{IN}, \mathit{OUT}, \mathit{DK}, \mathit{DC}\}$.*

Notation 3. *Given a four-state labelling L of $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$, $A \subseteq \mathcal{A}$ and $l \in \{\mathit{IN}, \mathit{OUT}, \mathit{DK}, \mathit{DC}\}$, we use $A \downarrow_l = \{a \in A \mid L(a) = l\}$ to restrict to arguments in A only labelled with l. We also denote with $L \downarrow_A$ a total mapping $L \downarrow_A$: $A \rightarrow$ {IN, OUT, DK, DC}.*

We see in Figure 5 an example of four-state weighted labelling. Accepted and rejected arguments, labelled with IN and OUT as usual, are still highlighted in green and red, respectively. An argument with label DK, which is highlighted in yellow, could be both accepted and rejected, meaning that we cannot decide about its acceptability (we "don't know", indeed). The DC label is depicted in grey and stands for "don't care" [6] and identifies arguments that are not interesting

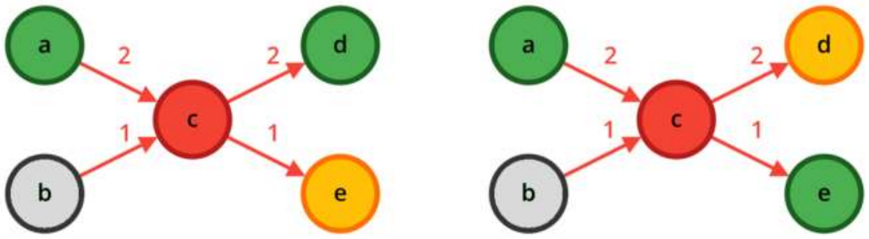


Figure 5: Two possible labellings of a WAF$_{\mathbb{S}}$ with $\mathbb{S} = \mathbb{S}_{weighted}$.

to analyse and that we just want to ignore. Finally, arguments in $\mathcal{U} \setminus \mathcal{A}$ (that are only part of the universe but not of the AF) are not labelled.

According to the definition of collective weighted defence (Definition 10), a set of arguments is defended from an attacker c only if the $\bigotimes$ of all the defending arguments is stronger than the $\bigotimes$ of the attacks coming from c. This means that the strength of the attacks of the defending arguments is distributed among the defended arguments and it is not guaranteed for two arguments that are separately w-defended to still be w-defended when considered together (this is what happens in the example of Figure 5 with arguments d and e).

We give a characterisation of four-state weighted semantics through the notion of labelling of WAF$_{\mathbb{S}}$ following the intuition that attacks of defending arguments are "consumed" by the defended one. In particular, an argument that cannot be accepted because its defenders are not strong enough will be labelled UNDEC. The first semantics we investigate is the basic requirement of conflict-freeness.

Fact 1 (w-conflict-free four-state labelling). *The w-conflict-free four-state labelling coincides with the w-conflict-free labelling.*

We want to identify a set of non-conflicting arguments, so we don't have to consider the weight of the attacks, but only if attacks exist between arguments in this set. We now define the w-admissible four-state labelling.

Definition 15 (w-admissible four-state labelling). *Let L be a four-state labelling of a WAF$_{\mathbb{S}}$ $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ and $a \in \mathcal{A}$. L is w-admissible if and only if:*

- $L(a) = IN \implies$
  $(\forall b \in a^{-}.L(b) \in \{OUT, DC\} \wedge L(b) = OUT \implies w_{b^{-}\downarrow_{IN}} \leq_{\mathbb{S}} w_{b^{+}\downarrow_{IN}})$
- $L(a) = OUT \iff w_{a^{-}\downarrow_{IN}} <_{\mathbb{S}} \top$

The condition $w_{b^{-}\downarrow_{\mathtt{IN}}} \leq_{\mathbb{S}} w_{b^{+}\downarrow_{\mathtt{IN}}}$ for IN arguments makes sure that defenders of a are stronger than the attack of b. For an argument to be OUT, then, we require $w_{a^{-}\downarrow_{\mathtt{IN}}} <_{\mathbb{S}} \top$, meaning that there must exist at least an attack coming from an IN argument. The two labellings in Figure 5 represent w-admissible four-state labellings for the considered WAF$_{\mathbb{S}}$.

Definition 16 (w-complete four-state labelling). *Let L be a four-state labelling of a WAF$_{\mathbb{S}}$ $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ and $a \in \mathcal{A}$. L is w-complete if and only if:*

- $L(a) = \texttt{IN} \iff$
  $(\forall b \in a^-. L(b) \in \{\texttt{OUT}, \texttt{DC}\} \wedge L(b) = \texttt{OUT} \implies w_{b^-\downarrow_{\texttt{IN}}} \leq_{\mathbb{S}} w_{b^+\downarrow_{\texttt{IN}}})$
- $L(a) = \texttt{OUT} \iff w_{a^-\downarrow_{\texttt{IN}}} <_{\mathbb{S}} \top$

A w-complete four-state labelling is also w-admissible. The difference is in the condition for IN arguments, which needs to be both necessary and sufficient. The four-state labellings in Figure 5 are not w-complete, since both have an UNDEC argument (e and d, respectively) which is only attacked by an OUT one.

Definition 17 (w-stable four-state labelling). *Let L be a four-state labelling of a WAF$_{\mathbb{S}}$ $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$. L is w-stable if and only if*

- *L is a w-complete four-state labelling and*
- *$\mathcal{A} \downarrow_{\texttt{DK}} = \emptyset$*

In contrast with the semantics in Definitions 15 and 16, a w-stable four-state labelling might not exist for a certain WAF$_{\mathbb{S}}$, depending on the presence of DK arguments. It is easy to verify that none of the labellings in Figure 5 is w-stable. We next present w-preferred and w-grounded four-state labelling for WAF$_{\mathbb{S}}$, which rely on the cardinality of the set of acceptable arguments.

Definition 18 (w-preferred labelling). *Let L be a four-state labelling of a WAF$_{\mathbb{S}}$ $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$. L is w-preferred if and only if*

- *L is a w-admissible four-state labelling and*
- *$\mathcal{A} \downarrow_{\texttt{IN}}$ is maximal among all the w-admissible four-state labellings*

Definition 19 (w-grounded four-state labelling). *Let L be a labelling of a WAF$_{\mathbb{S}}$ $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ and $a \in \mathcal{A}$. L is w-grounded if and only if:*

- *$L(a) = \texttt{IN} \iff$ for all w-complete four-state labellings L', $L'(a) = \texttt{IN}$ and*
- *$L(a) = \texttt{OUT} \iff w_{a^-\downarrow_{\texttt{IN}}} <_{\mathbb{S}} \top$*

We summarize in Table 1 the conditions given for the presented labellings. Next, we show how four-state labelling-based semantics for WAF$_{\mathbb{S}}$ can be traced to their three-state counterparts.

Theorem 1. *L is a w-conflict-free four-state labelling on $F = \langle \mathcal{A}, \mathcal{R} \rangle$ if and only if $L \downarrow_{\mathcal{A}}$ is a w-conflict-free three-state labelling and there exists a label renaming function such that, for all $a \in \mathcal{A}$, $(L(a) = \texttt{DC} \vee L(a) = \texttt{DK}) \implies L(a) = \texttt{UNDEC}$ and $L(a) = \texttt{UNDEC} \implies L(a) = \texttt{DC}$.*

Theorem 2. *L is a w-admissible (w-complete, w-stable, w-preferred, w-grounded) four-state labelling on $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$ with $\mathcal{A} \downarrow_{\texttt{DC}} = \emptyset$ if and only if $L \downarrow_{\mathcal{A}}$ is a w-admissible (w-complete, w-stable, w-preferred, w-grounded, respectively) three-state labelling and there exists a label renaming function such that, for all $a \in \mathcal{A}$, $L(a) = \texttt{DK} \iff L(a) = \texttt{UNDEC}$.*

Table 1
Summary of the labellings for WAF$_{\mathbb{S}}$.

Sem.	Conditions on `IN` arguments	Conditions on `OUT` arguments	Other
w-cf	$L(a) = \mathtt{IN} \implies a^- \downarrow_{\mathtt{IN}} = \emptyset$	$L(a) = \mathtt{OUT} \implies a^- \downarrow_{\mathtt{IN}} \neq \emptyset$	
w-adm	$L(a) = \mathtt{IN} \implies a^- = a^- \downarrow_{\{\mathtt{OUT},\mathtt{DC}\}}$ $\wedge \forall b \in a^- \downarrow_{\mathtt{OUT}} . w_{b^- \downarrow_{\mathtt{IN}}} \leq_{\mathbb{S}} w_{b^+ \downarrow_{\mathtt{IN}}}$	$L(a) = \mathtt{OUT} \iff w_{a^- \downarrow_{\mathtt{IN}}} <_{\mathbb{S}} \top$	
w-com	$L(a) = \mathtt{IN} \iff a^- = a^- \downarrow_{\{\mathtt{OUT},\mathtt{DC}\}}$ $\wedge \forall b \in a^- \downarrow_{\mathtt{OUT}} . w_{b^- \downarrow_{\mathtt{IN}}} \leq_{\mathbb{S}} w_{b^+ \downarrow_{\mathtt{IN}}}$	$L(a) = \mathtt{OUT} \iff w_{a^- \downarrow_{\mathtt{IN}}} <_{\mathbb{S}} \top$	
w-stb	$L(a) = \mathtt{IN} \iff a^- = a^- \downarrow_{\{\mathtt{OUT},\mathtt{DC}\}}$ $\wedge \forall b \in a^- . w_{b^- \downarrow_{\mathtt{IN}}} \leq_{\mathbb{S}} w_{b^+ \downarrow_{\mathtt{IN}}}$	$L(a) = \mathtt{OUT} \iff w_{a^- \downarrow_{\mathtt{IN}}} <_{\mathbb{S}} \top$	$\mathcal{A} \downarrow_{\mathtt{DK}} = \emptyset$
w-pre	$L(a) = \mathtt{IN} \implies a^- = a^- \downarrow_{\{\mathtt{OUT},\mathtt{DC}\}}$ $\wedge \forall b \in a^- . w_{b^- \downarrow_{\mathtt{IN}}} \leq_{\mathbb{S}} w_{b^+ \downarrow_{\mathtt{IN}}}$	$L(a) = \mathtt{OUT} \iff w_{a^- \downarrow_{\mathtt{IN}}} <_{\mathbb{S}} \top$	$\mathcal{A} \downarrow_{\mathtt{IN}}$ is max w-adm
w-gde	$L(a) = \mathtt{IN} \iff \forall L' w$-com. $L'(a) = \mathtt{IN}$	$L(a) = \mathtt{OUT} \iff w_{a^- \downarrow_{\mathtt{IN}}} <_{\mathbb{S}} \top$	

The intuition behind Theorem 2 is that the acceptability of all labelled arguments in a WAF$_{\mathbb{S}}$ (that is, those labelled by $L \downarrow_{\mathcal{A}}$) must depend only on the state of arguments that are not ignored. The proof is carried out by comparing Definition 13 with the conditions given for the four-state case. Moreover, since the four-state labelling introduced in this paper generalises the three-state one [14], we obtain a direct correspondence with weighted extensions.

Theorem 3. *Let L^F be a four-state labelling on $F = \langle \mathcal{A}, \mathcal{R}, W, \mathbb{S} \rangle$. L^F is a w-conflict-free labelling if and only if $\mathcal{A} \downarrow_{IN}$ is a w-conflict-free extension of F. Moreover, L^F is a w-admissible (respectively w-complete, w-stable, w-preferred, w-grounded) four-state labelling if and only if $\mathcal{A} \downarrow_{IN}$ is a w-admissible (respectively w-complete, w-stable, w-preferred, w-grounded) extension of $F' = \langle \mathcal{A} \downarrow_{\{IN,OUT,DK\}}, \mathcal{R} \downarrow_{\{IN.OUT,DK\}} \rangle$.*

Finally, we observe that any four-state weighted labelling instantiated with a boolean c-semiring corresponds to a four-state labelling. Indeed, when a WAF$_{\mathbb{S}}$ is instantiated with a boolean c-semiring, all the attacks in the framework are associated with the value $false$ and $w_{a^- \downarrow_{\mathtt{IN}}}$ always corresponds to $false$ if a has at least one attacker.

Theorem 4. *Let F be a WAF$_{\mathbb{S}}$ where $\mathbb{S}$ is a boolean c-semiring. If L is a w-admissible (respectively w-complete, w-stable, w-preferred, w-grounded) four-state labelling of F, then L is also an admissible (respectively complete, stable, preferred, grounded) four-state labelling.*

4. Related Work

The problem of extending classical AFs with values expressing the strength of arguments and attacks is widely studied, and many different approaches have been presented in the literature. Amgoud and Cayrol [25] take into account preference orderings for comparing arguments, while in a paper by Bench-Capon [26] the success of an attack conducted by an argument toward another one depends on an ordering among the "values" promoted by each argument.

A study on bipolar Weighted AFs is conducted by Pazienza et al. [27], who present an extension for weighted frameworks taking into account two different types of relations: one for attack and one for support. We consider, instead, Weighted AFs with only one type of possible relation between arguments (the attack relation). Note that there exist techniques for translating bipolar AFs into classical AFs [28], although the weighted case has not been investigated yet. Another formalism based on a notion of strength is given in a paper by Baroni et al. [29], where arguments in Quantitative Argumentation Debate Frameworks are evaluated through a scoring system. The main difference with our work lies in the fact that we take into account the basic definition of Weighted AFs [18], without further refinements on the framework level. Moreover, our study is focused on the interpretation of the labelling in the weighted case.

Labelling functions using four justification states are proposed by various authors [22, 6, 23]; the additional label identifies those arguments that should not be considered during the computation of acceptability. A more general labelling has also been proposed [13], which unifies different representations and can be mapped into sets of extensions. However, weights are not considered in any of these works.

For what concern the notion of weighted defence, many possible definitions can be considered: for instance, Martìnez et al. [10] use the relative strength of the attacks in order to determine if some defence constraints are satisfied, while Coste-Marquis et al. [9] aggregate the weights of the defence and check if this value is greater than the weight of the corresponding attack. On the other hand, we exploited the notion of collective weighted defence [7], which also generalises the other two approaches mentioned above.

5. Conclusion and Future Work

In this paper, we introduce labelling for Weighted AFs that uses up to four states to discern various grades of acceptability for arguments, namely `IN`, `OUT`, `DK` and `DC`. We also identify sets of conditions under which the proposed labelling corresponds to a weighted extension for some semantics. Our labelling function generalises both the classical approach for the non-weighted case and the three-state labelling for $\mathrm{WAF}_{\mathbb{S}}$.

The work can be expanded in many directions. In our setting, arguments only attacked by `DC` arguments are always labelled `IN`. Is future work, we want to consider a pessimistic interpretation for ignored arguments: since a `DC`-labelled argument a could be (re)considered into the AF, thus gaining an `IN`, `OUT` or `DK` label, arguments only attacked by a could be labelled `OUT` in turn. The definition we give of a four-state labelling-based semantics for Weighted AFs does not include conditions for `DK` arguments, since they are indirectly obtained from `IN` and `OUT`. In this sense, we would like to investigate the possible advantages of giving explicit conditions for labelling the `DK` arguments, similarly to what is by Modgil and Caminada [30] for classical AFs. We also plan to consider w-strongly admissible extensions [31, 14] and introduce the respective four-state labelling. In addition to the collective weighted defence [7] that we used in this paper, there are other notions of weighted defence [9, 10] that could be considered for obtaining different variations of the four-state weighted labelling. We would also like to take into account a relaxed version of the weighted defence [8] where two parameters (α and γ) are used to both enable a tolerance threshold for inconsistencies inside extensions and consider arguments that are not fully w-defended. Finally, extended versions of AFs (e.g., Bipolar Argumentation Frameworks [32]) could be investigated from the perspective of the four-state labelling-based semantics.

References

[1] J. Lawrence, J. Park, K. Budzynska, C. Cardie, B. Konat, C. Reed, Using argumentative structure to interpret debates in online deliberative democracy and eRulemaking, ACM Trans Internet Techn. 17 (2017) 25:1–25:22.

[2] D. Walton, M. Koszowy, Arguments from authority and expert opinion in computational argumentation systems, AI Soc. 32 (2017) 483–496.

[3] O. Cocarascu, F. Toni, Argumentation for machine learning: A survey, in: Computational Models of Argument - Proceedings of COMMA 2016, Potsdam, Germany, 12-16 September, 2016, volume 287 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2016, pp. 219–230.

[4] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-Person games, Artif. Intell. 77 (1995) 321–357.

[5] M. Caminada, On the Issue of Reinstatement in Argumentation, in: Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, September 13-15, 2006, Proceedings, volume 4160 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 111–123.

[6] H. Jakobovits, D. Vermeir, Robust semantics for argumentation frameworks, J. Log. Comput. 9 (1999) 215–261.

[7] S. Bistarelli, F. Rossi, F. Santini, A Collective Defence Against Grouped Attacks for Weighted Abstract Argumentation Frameworks, in: Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2016, AAAI Press, 2016, pp. 638–643.

[8] S. Bistarelli, F. Rossi, F. Santini, A novel weighted defence and its relaxation in abstract argumentation, Int J Approx Reason. 92 (2018) 66–86.

[9] S. Coste-Marquis, S. Konieczny, P. Marquis, M. A. Ouali, Weighted Attacks in Argumentation Frameworks, in: Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012, AAAI Press, 2012.

[10] D. C. Martínez, A. J. García, G. R. Simari, An Abstract Argumentation Framework with Varied-Strength Attacks, in: Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008, AAAI Press, 2008, pp. 135–144.

[11] S. Bistarelli, C. Taticchi, A labelling semantics for weighted argumentation frameworks, in: F. Calimeri, S. Perri, E. Zumpano (Eds.), Proceedings of the 35th Italian Conference on Computational Logic - CILC 2020, Rende, Italy, October 13-15, 2020, volume 2710 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 263–277.

[12] S. Bistarelli, C. Taticchi, Extending labelling semantics to weighted argumentation frameworks, in: E. Bell, F. Keshtkar (Eds.), Proceedings of the Thirty-Fourth International Florida Artificial Intelligence Research Society Conference, North Miami Beach, Florida, USA, May 17-19, 2021, 2021.

[13] S. Bistarelli, C. Taticchi, A unifying four-state labelling semantics for bridging abstract

argumentation frameworks and belief revision, in: C. S. Coen, I. Salvo (Eds.), Proceedings of the 22nd Italian Conference on Theoretical Computer Science, Bologna, Italy, September 13-15, 2021, volume 3072 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 93–106.

[14] S. Bistarelli, C. Taticchi, A labelling semantics and strong admissibility for weighted argumentation frameworks, J. Log. Comput. 32 (2022) 281–306.

[15] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowl. Eng Rev. 26 (2011) 365–410.

[16] S. Bistarelli, M. C. Meo, C. Taticchi, Timed concurrent language for argumentation, in: S. Monica, F. Bergenti (Eds.), Proceedings of the 36th Italian Conference on Computational Logic, Parma, Italy, September 7-9, 2021, volume 3002 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 1–15.

[17] S. Bistarelli, C. Taticchi, A concurrent language for argumentation, in: B. Fazzinga, F. Furfaro, F. Parisi (Eds.), Proceedings of the Workshop on Advances In Argumentation In Artificial Intelligence 2020 co-located with the 19th International Conference of the Italian Association for Artificial Intelligence (AIxIA 2020), Online, November 25-26, 2020, volume 2777 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 75–89.

[18] P. E. Dunne, A. Hunter, P. McBurney, S. Parsons, M. Wooldridge, Weighted argument systems: Basic definitions, algorithms, and complexity results, Artif. Intell. 175 (2011) 457–486.

[19] S. Bistarelli, F. Gadducci, Enhancing constraints manipulation in semiring-based formalisms, in: G. Brewka, S. Coradeschi, A. Perini, P. Traverso (Eds.), ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings, volume 141 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2006, pp. 63–67.

[20] S. Bistarelli, U. Montanari, F. Rossi, Semiring-based constraint satisfaction and optimization, J. ACM 44 (1997) 201–236.

[21] S. Bistarelli, F. Santini, A Hasse Diagram for Weighted Sceptical Semantics with a Unique-Status Grounded Semantics, in: Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), Lecture Notes in Computer Science, 2017.

[22] O. Arieli, On the acceptance of loops in argumentation frameworks, J. Log. Comput. 26 (2016) 1203–1234.

[23] R. Riveret, N. Oren, G. Sartor, A probabilistic deontic argumentation framework, Int. J. Approx. Reason. 126 (2020) 249–271.

[24] L. Amgoud, J. Ben-Naim, Ranking-based semantics for argumentation frameworks, in: W. Liu, V. S. Subrahmanian, J. Wijsen (Eds.), Scalable Uncertainty Management - 7th International Conference, SUM 2013, Washington, DC, USA, September 16-18, 2013. Proceedings, volume 8078 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 134–147.

[25] L. Amgoud, C. Cayrol, On the Acceptability of Arguments in Preference-Based Argumentation, in: UAI ’98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998, Morgan Kaufmann, 1998, pp. 1–7.

[26] T. J. M. Bench-Capon, Persuasion in Practical Argument Using Value-Based Argumentation

Frameworks, J Log Comput 13 (2003) 429–448.

[27] A. Pazienza, S. Ferilli, F. Esposito, Constructing and evaluating bipolar weighted argumentation frameworks for online debating systems, in: S. Bistarelli, M. Giacomin, A. Pazienza (Eds.), Proceedings of the 1st Workshop on Advances In Argumentation In Artificial Intelligence co-located with XVI International Conference of the Italian Association for Artificial Intelligence, AI^3@AI*IA 2017, Bari, Italy, November 16-17, 2017, volume 2012 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2017, pp. 111–125.

[28] G. Boella, D. M. Gabbay, L. W. N. van der Torre, S. Villata, Support in abstract argumentation, in: P. Baroni, F. Cerutti, M. Giacomin, G. R. Simari (Eds.), Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010, volume 216 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2010, pp. 111–122.

[29] P. Baroni, M. Romano, F. Toni, M. Aurisicchio, G. Bertanza, Automatic evaluation of design alternatives with quantitative argumentation, Argument Comput. 6 (2015) 24–49.

[30] S. Modgil, M. Caminada, Proof Theories and Algorithms for Abstract Argumentation Frameworks, in: Argumentation in Artificial Intelligence, Springer, 2009, pp. 105–129.

[31] P. Baroni, M. Giacomin, On principle-based evaluation of extension-based argumentation semantics, Artif. Intell. 171 (2007) 675–700.

[32] C. Cayrol, M. Lagasquie-Schiex, On the acceptability of arguments in bipolar argumentation frameworks, in: ECSQARU, volume 3571 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 378–389.

Modeling and Solving the Rush Hour puzzle[*]

Lorenzo Cian, Talissa Dreossi and Agostino Dovier

University of Udine, DMIF, Via delle Scienze 206, 33100 Udine, Italy

Abstract
We introduce the physical puzzle Rush Hour and its generalization. We briefly survey its complexity limits, then we model and solve it using declarative paradigms. In particular, we provide a constraint programming encoding in MiniZinc and a model in Answer Set Programming and we report and compare experimental results. Although this is simply a game, the kind of reasoning involved is the same that autonomous vehicles should do for exiting a garage. This shows the potential of logic programming for problems concerning transport problems and self-driving cars.

Keywords
Rush Hour, Planning, MiniZinc, ASP, Autonomous vehicles

1. Introduction

Rush Hour is a physical puzzle created by Nob Yoshigahara in 1970 and sold in USA for the first time in 1996. The game is played on a 6×6 board, on which there are a number of cars (of size 2) and trucks (of size 3). Cars and trucks can only move forwards or backwards (but not sideways). There is a unique exit door. The aim is to move the vehicles in such a way that the only red car can be driven out of the exit (see Figure 1 for an example).

The generalized rush hour problem, which has an arbitrary $m \times n$ grid size and allows to place the exit at any point on the perimeter of the grid, has been proved to be PSPACE-complete [1]. Due to this intrinsic limit we focus on the problem of finding a plan that allows to exit the red car with a fixed number t of moves. Then the solver will be run with $t = 1, 2, 3, \ldots$ until a solution (if any) is found.

Apart from [2, 1] where parameterized complexity is studied, in [3] the authors use model checking techniques for developing initial configurations that require high values for t making the instances *difficult*.

In [4] the authors studied the reasons why the transport puzzles are that complex, studying the sokoban, rush hour, and replacement puzzle. The complexity and a solution of sokoban in declarative programming was also presented in [5].

In this paper, as made in [6] and recently in [7] for other problems/puzzles, we model the

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

[*] Research partially supported by Fondazione Friuli/Università di Udine project on *Artificial Intelligence for Human Robot Collaboration* and by INDAM GNCS projects NoRMA and InSANE (CUP E55F22000270001).

agostino.dovier@uniud.it (A. Dovier)
www.dimi.uniud.it/dovier (A. Dovier)
0000-0003-2052-8593 (A. Dovier)

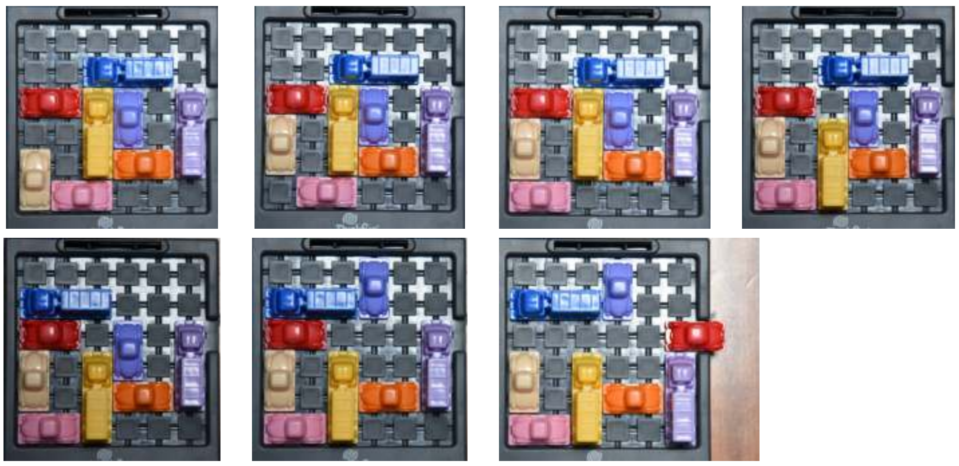


Figure 1: A six-moves exit plan for the red car on the physical game (the curious reader might find a plan with only five moves)

(generalization) of the rush hour puzzle in declarative programming using the language MiniZinc for a constraint programming encoding and Answer Set Programming for a logic programming encoding. We show the good results and the limits of the two approaches and set the basis for future development.

Although this is a game, self driving cars need to solve these kinds of puzzles for leaving a garage without damaging each other.

The paper is organized as follows: in Section 2 we set the background of the problem. We assume that the readers are aware of Constraint Programming and Answer Set Programming so we decided to avoid the definitions of those languages. The modeling in MiniZinc and ASP are presented in Sections 3 and 4, respectively. In Section 5 we report on the running time of the two approaches. Finally some conclusions are drawn in Section 6.

2. The problem and its complexity

A $m \times n$ board is a subset of the Cartesian plane identified by points $\mathcal{B} = \big\{(x, y) : 1 \leq x \leq m \land 1 \leq y \leq n\big\}$. Let us assume $(1, 1)$ is the bottom-left cell, and (m, n) the top-right cell. (x, y) is on the border of the grid if $x \in \{1, m\}$ or $y \in \{1, n\}$.

Let s be a function reporting the size of vehicles. A vehicle c can be of size $s(c) = 2$ (a car) or $s(c) = 3$ (a truck). A vechicle occupies exactly $s(c)$ adjacent cells. Given the position (x, y) of its front and its polar orientation north, south, east, west, the remaining cells occupied by the vehicle are univocally determined. For instance, if the orientation of a truck is toward south, the rest of the truck occupies $(x, y + 1), (x, y + 2)$.

A *garage* is a set $\mathcal{G} = \{(c_1, s_1), \ldots, (c_r, s_r)\}$ of pairs (c_i, s_i) where c_i is the name/index of a vehicle and $s_i = s(c_i) \in \{2, 3\}$ denotes its size.

An *allocation* of a garage in a $m \times n$ board is a set of triplets $\mathcal{T} = \{t_1, \ldots, t_r\}$ of the form $t_i = (x, y, o)$ where (x, y) is the grid cell occupied by the nose of the vehicle c_i and

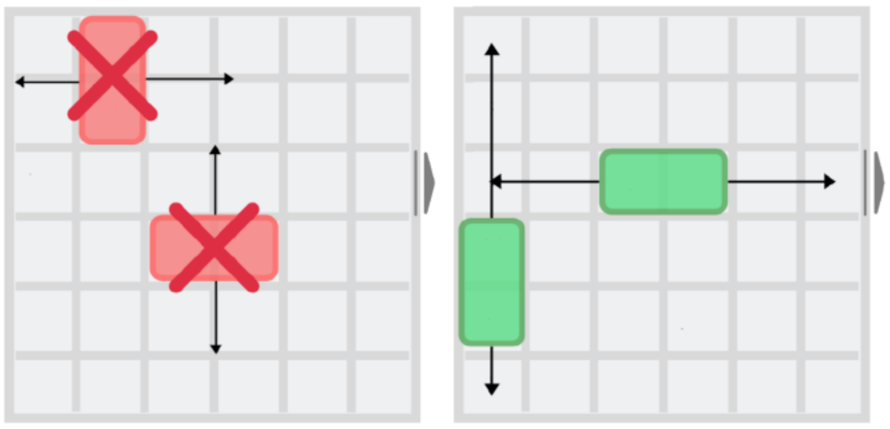


Figure 2: Allowed moves (right). The grey arrow denotes the exit gate $(6, 4)$

$o \in \{N, S, E, W\}$ (north, south, east, west, respectively) is its cardinal orientation, such that (1) all pieces of the vehicles are on the grid and (2) no pairs of them overlap.

Definition 2.1. *A* generalized rush hour *(briefly, GRH) instance is a tuple*

$$\langle \text{board-size}, \text{door}, \mathcal{G}, \mathcal{I} \rangle$$

where

- board-size *is a pair* $(m, n) \in \mathbb{N}^2$ *defining the grid size*
- door *is a pair* $(x_e, y_e) \in \mathbb{N}^2$ *on the border of the grid where the exit door is located*
- $\mathcal{G} = \{(c_1, s_1), \ldots, (c_r, s_r)\}$ *is a garage*
- $\mathcal{I} = \{t_1, \ldots, t_r\}$*, called the* initial state*, is an allocation of* $\mathcal{G}$

We assume that c_1 *identify the "red" car of the physical instance.*

Every vehicle c_i can be moved of one or more units, in one or the other direction consistent with its orientation o (see Fig 2). The car cannot exit from the board. If the vehicle moves of k units, the k cells must be free in the current state.

Definition 2.2. *Given a* generalized rush hour *(briefly, GRH) instance* $\langle \text{board-size}, \text{door}, \mathcal{C}, \mathcal{I} \rangle$ *a* plan *of length* ℓ *is a sequence of* ℓ *moves such that at the end a part of the vehicle* c_1 *occupies the door cell, and it is properly oriented to be allowed to exit the door.*

Let us observe that due to the kind of moves allowed, if $t_1 = (x_1, y_1, E)$ or $t_1 = (x_1, y_1, W)$ then $y_e = y_1$, and if $t_1 = (x_1, y_1, N)$ or $t_1 = (x_1, y_1, S)$ then $x_e = x_1$. If this does not holds then a plan cannot exist and the problem becomes trivial. Thus, we consider only instances that satisfy the above constraint.

As common in planning, there are two decision problems associated with GRH:

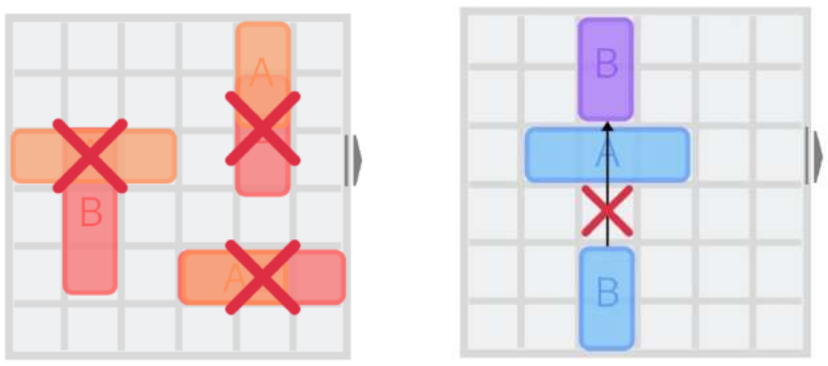


Figure 3: Vehicles cannot overlap, and cannot jump

1. Given an instance of GRH and $\ell \in \mathbb{N}$, establishing whether a a plan of lenght ℓ exists, and
2. Given an instance of GRH establishing if there is an $\ell \in \mathbb{N}$ such that a plan of lenght ℓ exists

Flake and Baum in [1] show how to encode Boolean formulas into instances of GRH proving NP-completeness of the former and PSPACE completeness of the latter.

Of course, the physical, 6×6 game has a finite number of possible instances, so, in principle it admits a constant time complexity using a program of huge size, storing features of all the possible instances. This size is of course not acceptable, thus we develop a program for GRH that, as particular case, solves 6×6 instances without making use of simplifications due to particular cases.

3. MiniZinc modeling

We describe our constraint programming encoding using the modeling language MiniZinc [8]. As common in planning we refer to a pair of garage (the set and kind of vehicles) $\mathcal{G}$ and their allocation $\mathcal{T}$ on a $m \times n$ grid as a *state*. We have to model states, actions, and the state change. The main constraints to be considered are the following:

- A vehicle cannot exit the board (neither completely nor partially)
- A vehicle cannot change its initial row or column or orientation
- Two different vehicles cannot overlap each other (see Figure 3)
- When the state is updated, a vehicle cannot jump over another (see Figure 3)

There are two main choices for the representation of a state:

- Focusing on the grid, namely defining a matrix B of size $m \times n$ where $B[i,j] = 0$ means that the cell is free and $B[i,j] = k$ that the cell is occupied by the vehicle k
- Focusing on the vehicles, namely using vectors of size r storing, in some way, the initial point and the orientation of all vehicles

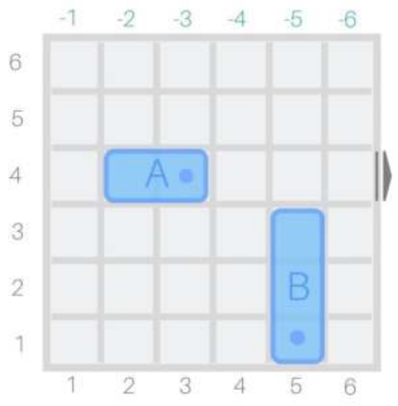


Figure 4: Example of representation: `size[A]=2`, `size[B]=3`, `versus[A]=4`, `versus[B]=-5`, `initial[A]=2`, `initial[B]=1`

Each representation has its pros and cons. For instance the matrix representation implements implicitly the non overlap constraint, while the vechicle representation uses less space and allows an easy update (only one vehicle per time-step). After the first empirical tests, we decided to focus on the second approach. Let us present it in some more detail.

For the sake of simplicity we'll use the standard board in what follows (i.e., $m = n = 6$). The encoding is easy to generalize.

The input consists in three arrays of length r. An array `size` stores for each vehicle its size (2 or 3). Changing direction of a vehicle is not possible. This means that once we know if it is horizontal (resp., vertical), the y coordinate (resp., x coordinate) is the same for all the computation. We store this info with a unique array `versus` that takes values in -6..6. If $\texttt{versus}[i] > 0$ then the vehicle is horizontal, and $\texttt{versus}[i]$ denotes its y coordinate (row). If $\texttt{versus}[i] < 0$ then the vehicle is vertical, and $\texttt{versus}[i]$ denotes its x coordinate (column). The GRH instance is completed by the array `initial` that fixes the other coordinates of each vehicle. For breaking symmetries, we do not store where the front of the vehicle is located. We store instead the smallest coordinate of the cells occupied by the vehicle (see Figure 4 for an example). Without loss of generality we assume that the red car is horizontal and that the exit door is located in the eastern cell of its row.

These were the static and input information. The dynamic behavior depends on two matrices that include the decision variables: `pos[i,j]` stores the smallest cell occuped by vehicle `i` at time `j`. `move[i,j]` is 0 if vehicle `i` does not move at time `j`, and $\delta \neq 0$ if it moves (positively or negatively) of δ positions. Although we don't need a matrix for the latter information (two vectors are sufficient) the matrix will allow an easy encoding of the inertia laws (as shown later).

The initial state can be stated as follows:

```
constraint
    forall(v in 1..vehicles)(pos[v,1]=initial[v]);
```

We will omit the declaration `constraint` before the following constraints.

The goal should be reached by a plan of exactly t time steps[1]

```
pos[1,t]=5;
```

The constraint stating that vehicles cannot exit the board is set in this way ($\texttt{pos}[\texttt{v},\texttt{s}] \geq 1$ is guaranteed by the domain of the variable):

```
forall(v in 1..vehicles, s in 1..steps)
     (pos[v,s]+size[v]-1<=6);
```

We need to state the non overlapping constraint. First we deal with pairs of vehicles in the same column or row:

```
forall(v1,v2 in 1..r,s in 1..t
         where (v1 < v2 /\ versus[v1] = versus[v2]))
    (pos[v1,s]+size[v1]-1 < pos[v2,s] \/
     pos[v2,s]+size[v2]-1 < pos[v1,s]);
```

Then we deal with pairs of hortogonal vehicles. In this case we explicitly avoid that they form a "cross"

```
 forall(v1,v2 in 1..r,s in 1..t
         where (versus[v1] > 0 /\ versus[v2] < 0))
    (not (pos[v1,s] <= -versus[v2] /\
        -versus[v2] <= pos[v1,s]+size[v1]-1  /\
        pos[v2,s] <= versus[v1]  /\
        versus[v1] <= pos[v2,s]+size[v2]-1));
```

Let us focus now on the moves. We need to state that there is exactly one move per time step.

```
forall(s in 1..t-1)
     (sum(v in 1..r)(move[v,s]!=0) = 1);
```

Other lower level, and slightly faster definitions have been tested, as well. The effect of a move action can be defined by this constraints. The fact that `move[v,s]` contains 0 for all vehicles v but one allows us to easily deal with inertia.

```
forall(v in 1..r, s in 1..t-1)
      (pos[v,s+1] = pos[v,s] + move[v,s]);
```

It remains to state that cars cannot jump during the move. This can be made as follows. For jumps on vehicles in the same row/column:

```
forall(s in 1..t-1, v1,v2 in 1..r
        where  ((v1<v2) /\ versus[v1]=versus[v2]))(
    not (pos[v1,s]<=pos[v2,s] /\ pos[v1,s+1] > pos[v2,s+1]) /\
    not (pos[v2,s]<=pos[v1,s] /\ pos[v2,s+1] > pos[v1,s+1]));
```

[1]Or alternatively, of at most t steps by defining a variable `min` as `var 1..t: min` and requiring `pos[1,min]=5`.

And for vertical and horizontal jumps on orthogonal cars

```
forall(s in 1..t-1, v1,v2 in 1..r
        where (versus[v1] < 0 /\ versus[v2] > 0))(
    (pos[v2,s] <= -versus[v1] /\
    -versus[v1] <= pos[v2,s]+size[v2]-1)
 -> (pos[v1,s] < versus[v2] -> pos[v1,s+1] < versus[v2]) /\
    (pos[v1,s] > versus[v2] -> pos[v1,s+1] > versus[v2]));

forall(s in 1..t-1, v1,v2 in 1..r
        where (versus[v1] > 0 /\ versus[v2] < 0))(
    (pos[v2,s] <= versus[v1] /\
     versus[v1] <= pos[v2,s]+size[v2]-1)
 -> (pos[v1,s] < -versus[v2] -> pos[v1,s+1] < -versus[v2]) /\
    (pos[v1,s] > -versus[v2] -> pos[v1,s+1] > -versus[v2]));
```

Finally, some symmetry breaking can be obtained by forbidding consecutive moves of the same vehicle:

```
forall(v in 1..r,s in 1..steps-2) (move[v,s] * move[v,s+1]=0);
```

4. Answer Set Programming Modeling

We developed two ASP models, one of them is based on the same ideas of the just described MiniZinc model. We explain below another approach that proved to be faster. As for the MiniZinc encoding we use the 6×6 grid, but the code is written in order to be easily generalizable. The code is tested with the ASP solvers clingo [9] and DLV [10].

First of all we set the grid size, the exit location and other domain predicates including the time range

```
grid(1..6, 1..6).
exit(6-1, 6/2 + 1).
move_amount(1..6).
direction(up; down; left; right).
time(0..t).
```

Vehicles are represented by facts of the kind

```
vehicle(Index, Size, Direction).
```

Where `Index` is the index (the name) of the car, `Size` is its size (2 or 3) and Direction states if it is `horizontal` or `vertical`, and its initial posizion is given as

```
position(Index, 0, X, Y).
```

where 0 stands for time 0, and X and Y are its initial coordinates. Precisely, if its an horizontal vehicle X is its minimal coordinate, if it is a vertical vehicle Y is its minimal coordinate (as made for the constraint modeling in the previous section).

We use intervals in the head of the rules to establish whether a grid cell is occupied or not:

```
busy(X, Y..Y+S-1, T) :- grid(X, Y), time(T),
         vehicle(A, S, vert), position(A, T, X, Y).

busy(X..X+S-1, Y, T) :- grid(X, Y), time(T),
         vehicle(A, S, horiz), position(A, T, X, Y).

free(X, Y, T) :- not busy(X, Y, T), grid(X, Y), time(T).
```

We use input allocations that do not overlap vehicles, however it would be simple checking consistency with a variation of the predicate busy. It is sufficient to add a parameter in the head and say that a cell is made busy by vehicle A and then requiring that it is impossible that a cell is made busy by two different vechicles. Similarly, we assume that the vehicles do not exit the board in the input allocations. These kind of constraints are instead controlled when actions are applied.

Let us set the executability conditions of a move:[2]

```
movable(A, T, up, N) :- grid(X,Y), grid(X,Y+S+N-1), time(T),
      vehicle(A, S, vert), position(A, T, X, Y),
      N {free(X, Y+S..Y+S+N-1, T)} N, move_amount(N).

movable(A, T, down, N) :- grid(X,Y), grid(X,Y-N), time(T),
      vehicle(A, S, vert), position(A, T, X, Y),
      N {free(X, Y-N..Y-1, T)} N, move_amount(N).

movable(A, T, left, N) :- grid(X,Y), grid(X-N,Y), time(T),
      vehicle(A, S, horiz), position(A, T, X, Y),
      N {free(X-N..X-1, Y, T)} N, move_amount(N).

movable(A, T, right, N) :- grid(X,Y), grid(X+S+N-1,Y), time(T),
      vehicle(A, S, horiz), position(A, T, X, Y),
      N {free(X+S..X+S+N-1, Y, T)} N, move_amount(N).
```

The four cases above are very similar: for a move of N steps, there must be N free cells in that direction. Let us observe how the aggregate is used in clause body.

Exactly one move per time is made:[3]

```
1 {move(A, T, D, N) : vehicle(A, S, D), direction(D),
                  movable(A, T, D, N), move_amount(N) } 1 :-
```

[2]The rules have been unfolded for N from 1 to 4 in the DLV encoding.
[3]The first rule was substituted with a choice rule and four constraints in the DLV encoding

```
      time(T).

moved(A, T) :- move(A, T, D, N), direction(D), move_amount(N).
```

The following rules compute the new position for moved and not moved vehicles:

```
position(A, T+1, X, Y+N) :-  move_amount(N) vtime(T), time(T+1),
      move(A, T, up, N), movable(A, T, up, N),
      vehicle(A, S, O), position(A, T, X, Y), grid(X, Y).

position(A, T+1, X, Y-N) :- move_amount(N) vtime(T), time(T+1),
      move(A, T, down, N), movable(A, T, down, N),
      vehicle(A, S, O), position(A, T, X, Y), grid(X, Y).

position(A, T+1, X-N, Y) :- move_amount(N) vtime(T), time(T+1),
      move(A, T, left, N), movable(A, T, left, N),
      vehicle(A, S, O), position(A, T, X, Y), grid(X, Y).

position(A, T+1, X+N, Y) :- move_amount(N) vtime(T), time(T+1),
      move(A, T, right, N), movable(A, T, right, N),
      vehicle(A, S, O), position(A, T, X, Y), grid(X, Y).

position(A, T+1, X, Y) :- grid(X, Y), time(T), time(T+1),
      not moved(A, T), position(A, T, X, Y),
      vehicle(A, S, O).
```

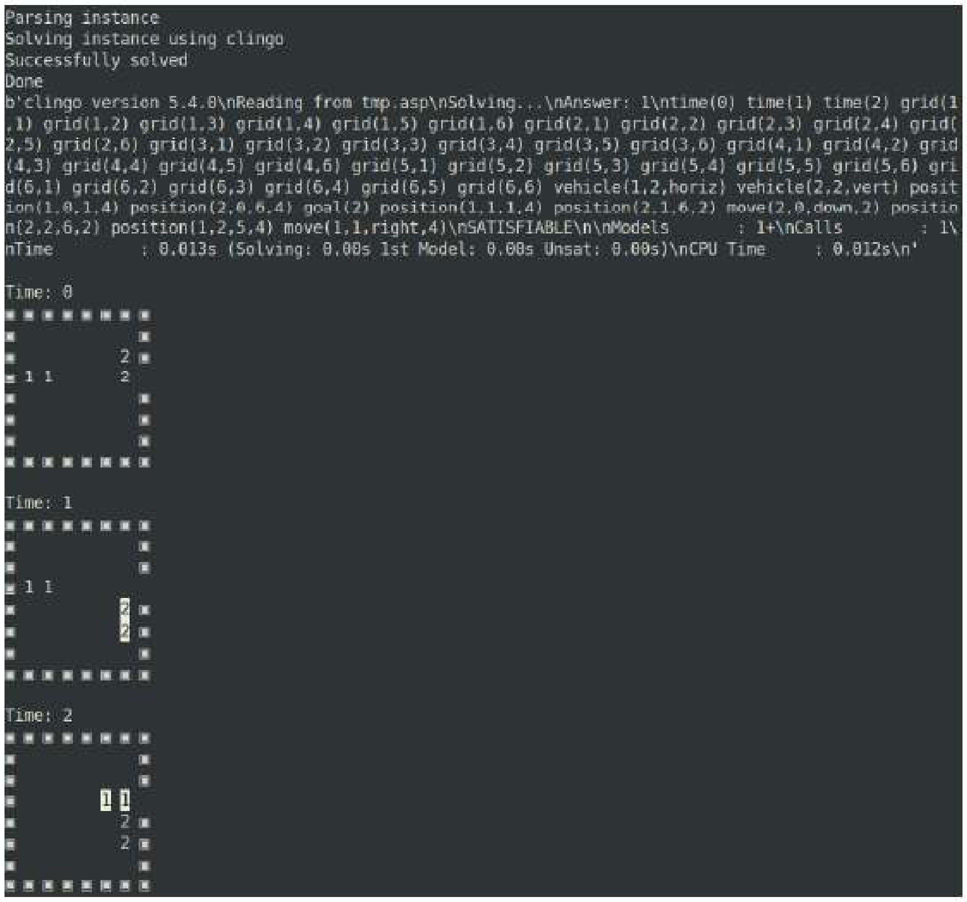
And finally we set the goal:

```
goal  :- position(1, t, X, Y), exit(X, Y).
:- not goal.
```

A Python interface has been written to call clingo and provide a graphical view of the plan. The input can be also given in command line using a string of chars. In the string, empty cells are represented by o, while vehicles are labeled by letters A, B, C, ... The 36 char string is obtained by storing the content of the rows, starting from the top one. The number of steps t is also passed. An example is reported in Figure 5.

5. Experimental Results

We compared the running time of the two proposed encodings on a set of benchmarks on the "official" 6×6 grid. Instances require increasing plan length. We run the codes on the minimum plan length leading to a solution. Tests are run on a system equipped with a AMD Ryzen 7 4700U CPU system, 16GB RAM, with OS 20.04 OS. We used version 2.5.5 of the MiniZinc to FlatZinc converter, the version 0.10.4 of the Chuffed solver [11], the version 5.4.0 of clingo, and the version 2.1.1 of DLV (for linux-x86_64). We set a timeout of 5 minutes.



```
Parsing instance
Solving instance using clingo
Successfully solved
Done
b'clingo version 5.4.0\nReading from tmp.asp\nSolving...\nAnswer: 1\ntime(0) time(1) time(2) grid(1
,1) grid(1,2) grid(1,3) grid(1,4) grid(1,5) grid(1,6) grid(2,1) grid(2,2) grid(2,3) grid(2,4) grid(
2,5) grid(2,6) grid(3,1) grid(3,2) grid(3,3) grid(3,4) grid(3,5) grid(3,6) grid(4,1) grid(4,2) grid
(4,3) grid(4,4) grid(4,5) grid(4,6) grid(5,1) grid(5,2) grid(5,3) grid(5,4) grid(5,5) grid(5,6) gri
d(6,1) grid(6,2) grid(6,3) grid(6,4) grid(6,5) grid(6,6) vehicle(1,2,horiz) vehicle(2,2,vert) posit
ion(1,0,1,4) position(2,0,6,4) goal(2) position(1,1,1,4) position(2,1,6,2) move(2,0,down,2) positio
n(2,2,6,2) position(1,2,5,4) move(1,1,right,4)\nSATISFIABLE\n\nModels       : 1+\nCalls        : 1\
nTime         : 0.013s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)\nCPU Time     : 0.012s\n'

Time: 0
■ ■ ■ ■ ■ ■ ■ ■
■             ■
■           2 ■
■ 1 1       2
■             ■
■             ■
■             ■
■ ■ ■ ■ ■ ■ ■ ■

Time: 1
■ ■ ■ ■ ■ ■ ■ ■
■             ■
■             ■
■ 1 1
■           2 ■
■           2 ■
■             ■
■ ■ ■ ■ ■ ■ ■ ■

Time: 2
■ ■ ■ ■ ■ ■ ■ ■
■             ■
■             ■
■         1 1
■           2 ■
■           2 ■
■             ■
■ ■ ■ ■ ■ ■ ■ ■
```

Figure 5: Example of the execution of the script:
`Python3.8 rush_hour.py "02 oooooooooooBAAoooBoooooooooooooooooo" lp`
Let us observe that the vehicle that just moved is highlighted

We used two benchmark sets. The first one was developed by us, it contains several instances of the physical game (there are cards with instances on them in the toy box) and other similar instances; globally it is a set of one hundred of instances from 5 to 17 steps. The second one is a set of 35 instances extracted from Michael Fogleman's database of Rush Hour configurations [12]. In this case, the plan length goes from 6 to 51 steps.

As far as the MiniZinc is concerned, we tested other solvers compatible with MiniZinc, namely Gecode version 6.3.0 and OR Tools version 9.3.10497. Both the solvers, with or without search annotations, performed considerably worse than Chuffed on simple instances, so we did not use them. The model without search annotations is the one which leads to the best performance with Chuffed. The default settings of both the MiniZinc compiler and Chuffed seem to be the ones which lead to the best performance. The default settings for clingo are also the ones that lead to the best performance.

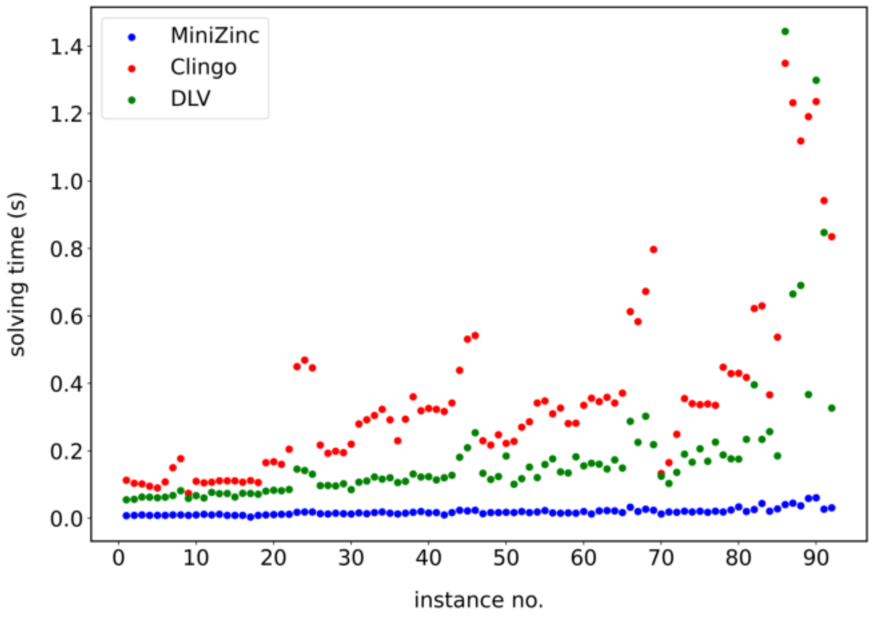


Figure 6: Comparison of the running time of the two encodings on a set of instances ordered by plan leght (maximum 17)

Both the approaches are sufficienlty efficient to solve all the instances of the first step of Figure 6 whitin the time limit, actually most of them in less than 0.2 seconds. Instead, with the second test, it can be oberved that the constraint modeling scales better as the number of steps increases. The problems arise when the plan length is more than 30. We have noticed that it is not simply a grounding problem, since for the most difficult instances (plan length 51), the grounded file has 71K lines, with a size in the text format of 6 MB, still not an issue. By the way, the solution in this case is found in 20 minutes (the timeout was set to 5 minutes). On these instances, with the default settings, DLV performs slightly better than clingo.

6. Future work and conclusions

We have presented two declarative encodings of the Rush Hour transport puzzle. Both of them are written using declarative code, without particular optimizations. The Mininizinc code, also thanks to the efficiency of the solver Chuffed is capable of solving hard instances in less than one second. The ASP code is extremely fast for plan lengths less than 30. Then solving takes more time, in any case within 20 minutes.

As future work, we would like to experiment the whole set of tests of Fogleman [12] (we have used only a sampling of it) and the whole set of instances of the physical game (printed on cards sold with the toy). We will embed some domain heuristics [13, 14] and adding a graphical

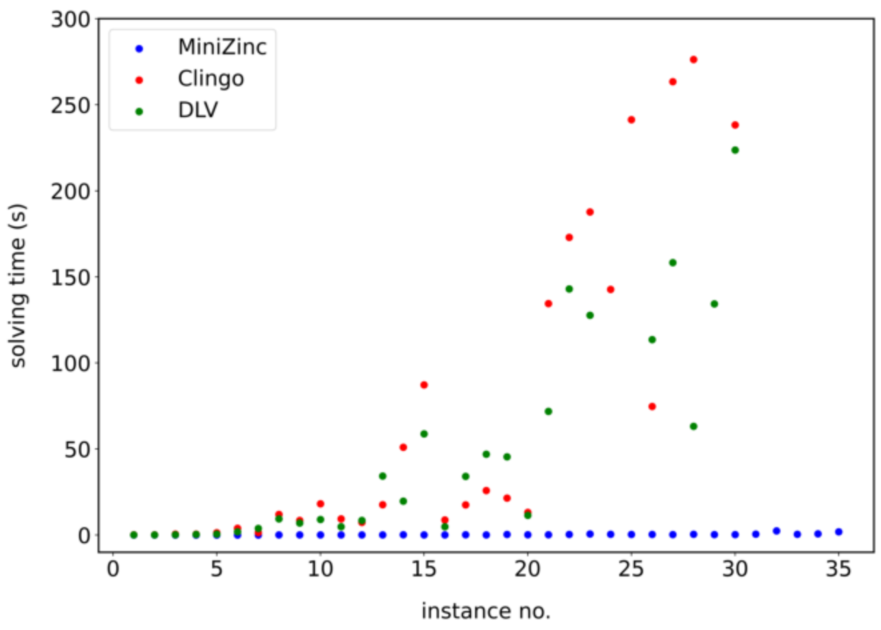


Figure 7: Comparison of the running time of the two encodings on a set of 35 instances (plan lenght from 6 to 51, timeout 5 minutes)

interface for generating the input and for the animation of the solutions.

Moreover, in order to add some realism to the game, we would like to admit cars and trucks to turn right/left of 90°. Another interesting aspects would be the one of a multiagent systems where more cars can move in parallel.

The codes are written almost completely in the paper, however, we will report them together with the set of instances in http://clp.dimi.uniud.it/sw/.

References

[1] G. W. Flake, E. B. Baum, Rush hour is pspace-complete, or "why you should generously tip parking lot attendants", Theor. Comput. Sci. 270 (2002) 895–911. doi:10.1016/S0304-3975(01)00173-6.

[2] H. Fernau, T. Hagerup, N. Nishimura, P. Ragde, K. Reinhardt, On the parameterized complexity of the generalized rush hour puzzle, in: Proceedings of the 15th Canadian Conference on Computational Geometry, CCCG'03, Halifax, Canada, August 11-13, 2003, 2003, pp. 6–9. URL: http://www.cccg.ca/proceedings/2003/22.pdf.

[3] S. Collette, J. Raskin, F. Servais, On the symbolic computation of the hardest configurations of the RUSH HOUR game, in: H. J. van den Herik, P. Ciancarini, H. H. L. M. Donkers (Eds.), Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31,

2006. Revised Papers, volume 4630 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 220–233. doi:10.1007/978-3-540-75538-8_20.

[4] P. Jarusek, R. Pelánek, What determines difficulty of transport puzzles?, in: R. C. Murray, P. M. McCarthy (Eds.), Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, May 18-20, 2011, Palm Beach, Florida, USA, AAAI Press, 2011. URL: http://aaai.org/ocs/index.php/FLAIRS/FLAIRS11/paper/view/2518.

[5] N. Zhou, A. Dovier, A tabled prolog program for solving sokoban, Fundam. Informaticae 124 (2013) 561–575. doi:10.3233/FI-2013-849.

[6] A. Dovier, A. Formisano, E. Pontelli, An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems, J. Exp. Theor. Artif. Intell. 21 (2009) 79–121. doi:10.1080/09528130701538174.

[7] N. Rizzo, A. Dovier, 3cosoku and its declarative modeling, J. Log. Comput. 32 (2022) 307–330. doi:10.1093/logcom/exab086.

[8] P. J. Stuckey, K. Marriott, G. Tack, The minizinc handbook, 2022. URL: https://www.minizinc.org/.

[9] University of Potsdam, Potassco, the potsdam answer set solving collection, 2022. URL: https://potassco.org/.

[10] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri, J. Zangari, The ASP system DLV2, in: M. Balduccini, T. Janhunen (Eds.), Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings, volume 10377 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 215–221. doi:10.1007/978-3-319-61660-5_19.

[11] G. Chu, M. G. de la Banda, C. Mears, P. J. Stuckey, Symmetries, almost symmetries, and lazy clause generation, Constraints An Int. J. 19 (2014) 434–462. doi:10.1007/s10601-014-9163-9.

[12] M. Fogleman, Solving rush hour, the puzzle, 2022. URL: https://www.michaelfogleman.com/rush/.

[13] M. Gebser, B. Kaufmann, J. Romero, R. Otero, T. Schaub, P. Wanko, Domain-specific heuristics in answer set programming, in: M. desJardins, M. L. Littman (Eds.), Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA, AAAI Press, 2013. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6278.

[14] C. Dodaro, P. Gasteiger, N. Leone, B. Musitsch, F. Ricca, K. Schekotihin, Combining answer set programming and domain heuristics for solving hard industrial problems (application paper), Theory Pract. Log. Program. 16 (2016) 653–669. doi:10.1017/S1471068416000284.

From Weighted Conditionals with Typicality to a Gradual Argumentation Semantics and back (Extended Abstract)

Laura Giordano

DISIT - Università del Piemonte Orientale, Viale Michel 11, I-15121, Alessandria, Italy

Abstract
A fuzzy multi-preferential semantics has been recently proposed for weighted conditional knowledge bases with typicality, and used to develop a logical semantics for Multilayer Perceptrons, by regarding a deep neural network (after training) as a weighted conditional knowledge base. Based on different variants of this semantics, we propose some new gradual argumentation semantics, and relate them to the family of the gradual semantics. This also suggests an approach for defeasible reasoning over a weighted argumentation graph, building on the proposed semantics.

This extended abstract reports about some work [24] investigating the relationships between the weighted conditional knowledge bases with typicality, under a fuzzy semantics, and gradual argumentation semantics [17, 36, 21, 22, 2, 5, 3, 46], and discusses some extension of this work in the direction of allowing defeasible reasoning over weighted argumentation graphs [26].

Argumentation is a reasoning approach which, in its different formulations and semantics, has been used in different contexts in the multi-agent setting, from social networks [42] to classification [4], and it is very relevant for decision making and for explanation [47]. The argumentation semantics are strongly related to other non-monotonic reasoning formalisms and semantics [20, 1].

Our starting point in this work is a preferential semantics for commonsense reasoning which has been proposed for a description logic with typicality. Preferential description logics have been studied in the last fifteen years to deal with inheritance with exceptions in ontologies, based on the idea of extending the language of Description Logics (DLs) by allowing for non-strict forms of inclusions, called *typicality or defeasible inclusions*, of the form $\mathbf{T}(C) \sqsubseteq D$ (meaning "the typical C-elements are D-elements" or "normally C's are D's"), with different preferential semantics [28, 13] and closure constructions [15, 14, 29, 8, 44, 16, 27]. Such defeasible inclusions correspond to Kraus, Lehmann and Magidor (KLM) conditionals $C \mathrel{|\!\sim} D$ [40, 41], and defeasible DLs inherit and extend some of the preferential semantics and closure constructions developed within preferential and conditional approaches to commonsense reasoning by Kraus, Lehmann and Magidor [40], Pearl [43], Lehmann [41], Geffner and Pearl [23], Benferhat et al. [7].

In previous work [33], a concept-wise multi-preferential semantics for weighted conditional knowledge bases (KBs) has been proposed to account for preferences with respect to different concepts, by allowing a set of typicality inclusions of the form $\mathbf{T}(C) \sqsubseteq D$ with positive or

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
laura.giordano@uniupo.it (L. Giordano)

negative weights, for some distinguished concepts C. The concept-wise multi-preferential semantics has been first introduced as a semantics for ranked DL knowledge bases [32], where conditionals are given a positive integer rank, and later extended to weighted conditional KBs, in the two-valued and in the fuzzy case, based on a different semantic closure construction in the spirit of Lehmann's lexicographic closure [41] and Kern-Isberner's c-representations [37, 38], but exploiting multiple preferences with respect to concepts.

The concept-wise multi-preferential semantics has been proven to have some desired properties from the knowledge representation point of view. In the two-valued case [32], it satisfies the KLM properties of a preferential consequence relation [40, 41], it allows to deal with specificity and irrelevance and avoids inheritance blocking or the "drowning problem" [43, 7], and deals with "ambiguity preservation" [23]. The plausibility of the concept-wise multi-preferential semantics has also been supported [30, 31] by showing that it is able to provide a logical interpretation to Kohonen' Self-Organising Maps [39], which are psychologically and biologically plausible neural network models. In the fuzzy case, the KLM properties of non-monotomic entailment have been studied in [25], showing that most KLM postulates are satisfied, depending on their reformulation and on the choice of fuzzy combination functions. It has been shown [33] that (both in the two-valued and in the fuzzy case) the multi-preferential semantics allows to describe the behavior of Multilayer Perceptrons (MLPs), after training, in terms of a preferential interpretation which, in the fuzzy case, can be proven to be a model (in a logical sense) of the weighted KB which is associated to the neural network.

The relationships between preferential and conditional approaches to non-monotonic reasoning and argumentation semantics are strong. Let us just mention, the work by Geffner and Pearl on Conditional Entailment, whose proof theory is defined in terms of "arguments" [23].

To investigate the relationships between the fuzzy multi-preferential semantics for weighted conditionals and gradual argumentation semantics [17, 36, 21, 22, 2, 5, 3, 46], we have introduced a new notion of φ-coherent fuzzy multi-preferential semantics [24] for weighted conditionals, besides the previously introduced notions of coherent [33] and faithful [25] fuzzy multi-preferential semantics. For weighted argumentation graphs, where positive and negative weights can be associated to pairs of arguments, we have proposed three new gradual semantics (namely, a coherent, a faithful and a φ-coherent semantics) inspired by the fuzzy preferential semantics of weighted conditionals, and we have studied their relationships.

The relationship of the φ-coherent semantics with the family of gradual semantics studied by Amgoud and Doder [2] has also been investigated, by slightly extending their gradual argumentation framework to deal with positive and negative weights to capture the strength of supports and of attacks. A correspondence between the gradual semantics based on a specific evaluation method M^{φ} and φ-coherent labelings has been proven [26]. Differently from the Fuzzy Argumentation Frameworks by Jenssen et al. [36], where an attack relation is a fuzzy binary relation over the set of arguments, here we have considered real-valued weights associated to pairs of arguments.

While in [33] a deep neural network (possibly containing cycles) is mapped to a weighted conditional knowledge base, a deep neural network can as well be seen as a weighted argumentation graph, with positive and negative weights, under the proposed semantics. In this view, φ-coherent labelings correspond to stationary states of the network (where each unit in the network is associated to an argument and the activation value of the unit can be regarded as the weight of the corresponding argument). This is in agreement with previous work on the relationship between argumentation frameworks and neural networks, first investigated by Garcez, Gabbay and Lamb [19] and recently by Potyca [45].

The work by Garcez, et al. [19] combines value-based argumentation frameworks [6] and neural-symbolic learning systems by providing a translation from argumentation networks to neural networks with 3 layers (input, output layer and one hidden layer). This enables the accrual of arguments through learning as well as the parallel computation of arguments. The work by Potyca [45] considers a quantitative bipolar argumentation frameworks (QBAFs) similar to [5] and exploits an *influence function* based on the logistic function to define an MLP-based semantics σ_{MLP} for a QBAF. The paper studies convergence conditions both in the discrete and in the continuous case, as well as the semantic properties of MLP-based semantics, and proves that all properties for the QBAF semantics proposed in [2] are satisfied. On the other hand, as shown in [26], the φ-coherent model semantics fails to satisfy some of the properties in [2].

The strong relationships between the semantics for weighted conditionals and gradual argumentation semantics also leads to an approach for defeasible reasoning over a weighted argumentation graphs, building on φ-coherent labelings to evaluate conditional properties of the argumentation graph [26]. In essence, a multi-preferential model can be constructed over a (finite) set of φ-labelling Σ, which allows (fuzzy) conditional formulas over arguments to be validated by model checking over a preferential model. This would, for instance, allow to verify properties like: "does normally argument A_2 follows from argument A_1 with a degree greater than 0.7?" This query can be formalized by a fuzzy inclusion $\mathbf{T}(A_1) \sqsubseteq A_2 > 0.7$, similarly to those considered for weighted knowledge bases. This approach has been exploited for the verification of defeasible properties of Multilayer Perceptrons [34]. Whether this approach can be extended to the other gradual semantics, and under which conditions on the evaluation method, requires further investigation for future work.

Observe also that, in a weighted conditional knowledge base, the concepts C and D occurring in a typicality inclusion $\mathbf{T}(C) \sqsubseteq D$ are not required to be concept names, but can be complex concepts. In particular, in the boolean fragment $\mathcal{LC}$ of $\mathcal{ALC}$, D can be any boolean combination of concept names. The correspondence between weighted attacks/supports (A_i, A_j) in the argumentation graph G and weighted conditionals $\mathbf{T}(A_i) \sqsubseteq A_j$ suggests a possible generalization of the structure of the weighted argumentation graph by allowing attacks/supports by a boolean combination of arguments. The labelling of arguments in the set $[0, 1]$ can indeed be extended to boolean combinations of arguments using the fuzzy combination functions, as for boolean concepts in the conditional semantics (e.g., by letting $\sigma(A_1 \wedge A_2) = min\{\sigma(A_1), \sigma(A_2)\}$, using the minimum t-norm as in Zadeh fuzzy logic). This also relates to the work considering “sets of attacking (resp. supporting) arguments”; i.e., several argument together attacking (or supporting) an argument. Indeed, for gradual semantics, the sets of attacking arguments framework (SETAF) has been studied by Yun and Vesic [46], by considering “the force of the set of attacking (resp. supporting) arguments to be the force of the weakest argument in the set" [46]. This would correspond to interpret the set of arguments as a conjunction, using minimum t-norm.

The correspondence between Abstract Dialectical Frameworks [12] and Nonmonotonic Conditional Logics has been studied by Heyninck, Kern-Isberner and Thimm [35], with respect to the two-valued models, the stable, the preferred semantics and the grounded semantics of ADFs. Whether the coherent/faithful/φ-coherent semantics developed in the paper for weighted argumentation (as well as their two-valued and many-valued variants) can be reformulated for a (weighted) Abstract Dialectical Frameworks, and which are the relationships with the work in [35], also requires investigation for future work.

Undecidability results for fuzzy description logics with general inclusion axioms (e.g., by Cerami and Straccia [18] and by Borgwardt and Peñaloza [9]) motivate restricting the logics to finitely valued semantics [10], and the investigation of decidable approximations of fuzzy multi-preferential entailment, under the different semantics. An ASP approach for reasoning under finitely multi-valued fuzzy semantics for weighted conditional knowledge bases has been proposed in [34], by exploiting *asprin* [11] for defeasible reasoning through the computation of preferred answer sets. As a proof of concept, this approach has been experimented for checking properties of some trained Multilayer Perceptrons. A similar investigation of the two-valued and many-valued case might also be of interest for the semantics of weighted argumentation graphs introduced in this work.

Acknowledgement: This research is partially supported by INDAM-GNCS Project 2022 "LESLIE: LogichE non-claSsiche per tooL Intelligenti ed Explainable".

References

[1] G. Alfano, S. Greco, F. Parisi, and I. Trubitsyna. On the semantics of abstract argumentation frameworks: A logic programming approach. *TPLP*, 20(5):703–718, 2020.

[2] L. Amgoud, J. Ben-Naim, D. Doder, and S. Vesic. Acceptability semantics for weighted argumentation frameworks. In *IJCAI 2017, Melbourne, Australia*, pages 56–62, 2017.

[3] L. Amgoud and D. Doder. Gradual semantics accounting for varied-strength attacks. In *Proceedings AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, pages 1270–1278, 2019.

[4] L Amgoud and M. Serrurier. Agents that argue and explain classifications. *Auton. Agents Multi Agent Syst.*, 16(2):187–209, 2008.

[5] P. Baroni, A. Rago, and F. Toni. How many properties do we need for gradual argumentation? In *Proc. AAAI 2018*, pages 1736–1743, 2018.

[6] T. J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13(3):429–448, 2003.

[7] S. Benferhat, C. Cayrol, D. Dubois, J. Lang, and H. Prade. Inconsistency management and prioritized syntax-based entailment. In *Proc. IJCAI'93, Chambéry,*, pages 640–647, 1993.

[8] P. A. Bonatti and L. Sauro. On the logical properties of the nonmonotonic description logic DL^{N}. *Artif. Intell.*, 248:85–111, 2017.

[9] S. Borgwardt and R. Peñaloza. Undecidability of fuzzy description logics. In Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, editors, *Proc. KR 2012, Rome, Italy, June 10-14, 2012.* AAAI Press, 2012.

[10] S. Borgwardt and R. Peñaloza. The complexity of lattice-based fuzzy description logics. *J. Data Semant.*, 2(1):1–19, 2013.

[11] G. Brewka, J. P. Delgrande, J. Romero, and T. Schaub. asprin: Customizing answer set preferences without a headache. In *Proc. AAAI 2015*, pages 1467–1474, 2015.

[12] G Brewka, H. Strass, S. Ellmauthaler, J. P. Wallner, and S. Woltran. Abstract dialectical frameworks revisited. In *Proc. IJCAI 2013, pages 803–809, 2013.*

[13] K. Britz, J. Heidema, and T. Meyer. Semantic preferential subsumption. In G. Brewka and J. Lang, editors, *KR 2008*, pages 476–484, Sidney, Australia, September 2008. AAAI Press.

[14] G. Casini, T. Meyer, I. J. Varzinczak, , and K. Moodley. Nonmonotonic Reasoning in Description Logics: Rational Closure for the ABox. In *26th Int. Workshop on Description Logics (DL 2013)*, volume 1014 of *CEUR Workshop Proceedings*, pages 600–615, 2013.

[15] G. Casini and U. Straccia. Rational Closure for Defeasible Description Logics. In T. Janhunen and I. Niemelä, editors, *JELIA 2010*, volume 6341 of *LNCS*, pages 77–90, Helsinki, Sept. 2010. Springer.

[16] G. Casini, U. Straccia, and T. Meyer. A polynomial time subsumption algorithm for nominal safe elo$\perp$ under rational closure. *Inf. Sci.*, 501:588–620, 2019.

[17] C. Cayrol and M. Lagasquie-Schiex. Graduality in argumentation. *J. Artif. Intell. Res.*, 23:245–297, 2005.

[18] M. Cerami and U. Straccia. On the undecidability of fuzzy description logics with gcis with lukasiewicz t-norm. *CoRR*, abs/1107.4212, 2011.

[19] A. S. d'Avila Garcez, D. M. Gabbay, and L. C. Lamb. Value-based argumentation frameworks as neural-symbolic learning systems. *J. Log. Comput.*, 15(6):1041–1058, 2005.

[20] P. M. Dung. On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77:321–357, 1995.

[21] P. E. Dunne, A. Hunter, P. McBurney, S. Parsons, and M. J. Wooldridge. Weighted argument systems: Basic definitions, algorithms, and complexity results. *Artif. Intell.*, 175(2):457–486, 2011.

[22] S. Egilmez, J. G. Martins, and J. Leite. Extending social abstract argumentation with votes on attacks. In *TAFA 2013, Beijing, China, Aug. 3-5, LNCS 8306*, pages 16–31. Springer, 2013.

[23] Hector Geffner and Judea Pearl. Conditional entailment: Bridging two approaches to default reasoning. *Artif. Intell.*, 53(2-3):209–244, 1992.

[24] L. Giordano. From weighted conditionals of multilayer perceptrons to a gradual argumentation semantics. In *5th Workshop on Advances in Argumentation in Artificial Intelligence 2021, co-located with AIxIA 2021, Milan, Italy, November 29, 2021*, volume 3086 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.

[25] L. Giordano. On the KLM properties of a fuzzy DL with Typicality. In *16th European Conf. on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU 2021, Prague, Sept. 21-24*, volume 12897 of *LNCS*, pages 557–571. Springer, 2021.

[26] L. Giordano. From weighted conditionals with typicality to a gradual argumentation semantics and back. In *20th Workshop on Non-Monotonic Reasoning, NMR 2022, Haifa, August 7-9*, 2022. To appear. A preliminary version in https://arxiv.org/abs/2110.03643.

[27] L. Giordano and V. Gliozzi. A reconstruction of multipreference closure. *Artif. Intell.*, 290, 2021.

[28] L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Preferential Description Logics. In *LPAR 2007*, volume 4790 of *LNAI*, pages 257–272, Yerevan, Armenia, October 2007. Springer.

[29] L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Semantic characterization of rational closure: From propositional logic to description logics. *Artif. Intell.*, 226:1–33, 2015.

[30] L. Giordano, V. Gliozzi, and D. Theseider Dupré. On a plausible concept-wise multipreference semantics and its relations with self-organising maps. In *CILC 2020, Rende, IT, Oct.*

13-15, 2020, volume 2710 of *CEUR*, pages 127–140, 2020.

[31] L. Giordano, V. Gliozzi, and D. Theseider Dupré. A conditional, a fuzzy and a probabilistic interpretation of self-organising maps. *Journal of Logic and Computation, https://doi.org/10.1093/logcom/exab082*, 2022.

[32] L. Giordano and D. Theseider Dupré. An ASP approach for reasoning in a concept-aware multi-preferential lightweight DL. *Theory Pract. Log. Program.*, 20(5):751–766, 2020.

[33] L. Giordano and D. Theseider Dupré. Weighted defeasible knowledge bases and a multipreference semantics for a deep neural network model. In *Proc. JELIA 2021, May 17-20*, volume 12678 of *LNCS*, pages 225–242. Springer, 2021. An extended version in https://arxiv.org/abs/2012.13421v2.

[34] L. Giordano and D. Theseider Dupré. An ASP approach for reasoning on neural networks under a finitely many-valued semantics for weighted conditional knowledge bases. *Theory Pract. Log. Program.*, 2022. https://www.doi.org/10.1017/S1471068422000163.

[35] J. Heyninck, G. Kern-Isberner, and M. Thimm. On the correspondence between abstract dialectical frameworks and nonmonotonic conditional logics. In *Proceedings of the Thirty-Third International Florida Artificial Intelligence Research Society Conference, May 17-20, 2020*, pages 575–580. AAAI Press, 2020.

[36] J. Janssen, M. De Cock, and D. Vermeir. Fuzzy argumentation frameworks. In *IPMU 2008*, pages 513–520, 2008.

[37] G. Kern-Isberner. *Conditionals in Nonmonotonic Reasoning and Belief Revision - Considering Conditionals as Agents*, volume 2087 of *LNCS*. Springer, 2001.

[38] G. Kern-Isberner and C. Eichhorn. Structural inference from conditional knowledge bases. *Stud Logica*, 102(4):751–769, 2014.

[39] T. Kohonen, M.R. Schroeder, and T.S. Huang, editors. *Self-Organizing Maps, Third Edition*. Springer Series in Information Sciences. Springer, 2001.

[40] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1-2):167–207, 1990.

[41] D. Lehmann and M. Magidor. What does a conditional knowledge base entail? *Artificial Intelligence*, 55(1):1–60, 1992.

[42] J. Leite and J. G. Martins. Social abstract argumentation. In *IJCAI 2011, Barcelona, Spain, July 16-22, 2011*, pages 2287–2292. IJCAI/AAAI, 2011.

[43] J. Pearl. System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In *TARK'90, Pacific Grove, CA, USA*, pages 121–135, 1990.

[44] M. Pensel and A. Turhan. Reasoning in the defeasible description logic $EL_{\perp}$ - computing standard inferences under rational and relevant semantics. *Int. J. Approx. Reasoning*, 103:28–70, 2018.

[45] N. Potyka. Interpreting neural networks as quantitative argumentation frameworks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, February 2-9, 2021*, pages 6463–6470. AAAI Press, 2021.

[46] B. Yun and S. Vesic. Gradual semantics for weighted bipolar setafs. In *Proc. ECSQARU 2021, Prague, Czech Republic, Sept. 21-24, 2021*, volume 12897 of *LNCS*, pages 201–214, 2021.

[47] Q. Zhong, X. Fan, X. Luo, and F. Toni. An explainable multi-attribute decision model based on argumentation. *Expert Syst. Appl.*, 117:42–61, 2019.

An ASP-based Approach to Master Surgical Scheduling

Linda Cademartori[1], Giuseppe Galatà[2], Carola Lo Monaco[1], Marco Maratea[1], Marco Mochi[1] and Marco Schouten[3]

[1] *University of Genoa, Genova, Italy*
[2] *SurgiQ srl, Genova, Italy*
[3] *KTH Royal Institute of Technology in Stockholm, Sweden*

Abstract
The problem of finding Master Surgical Schedules (MSS) consists of scheduling different specialties to the operating rooms of a hospital clinic. To produce a proper MSS, each specialty must be assigned to some operating rooms. The number of assignments is different for each specialty and can vary during the considered planning horizon. Realizing a satisfying schedule is of upmost importance for a hospital clinic. A poorly scheduled MSS may lead to unbalanced specialties availability and increase patients' waiting list, negatively affecting both the administrative costs of the hospital and the patient satisfaction. In this paper, we present a compact solution based on Answer Set Programming (ASP) to the MSS problem. We tested our solution on different scenarios: experiments show that our ASP solution provides satisfying results in short time, also when compared to other logic-based formalisms. Finally, we describe a web application we have developed for easy usage of our solution.

Keywords
Healthcare, Scheduling, Answer Set Programming

1. Introduction

Digital Health, defined as the usage of information and communication technologies in medicine and in the management processes of healthcare, arose several years ago, but has gained increasing importance in recent years. Thanks to new technologies and also due to new challenges such as an aging society, the COVID-19 pandemic and the need to reduce high costs. One of the major problems related to the modern hospitals are long waiting lists that reduce patients' satisfaction and the level of care offered to them. The Master Surgical Schedule (MSS) represents which specialty is assigned to each operating room in a particular day and session. The administrative practices of surgical departments, such as deciding which operating rooms are assigned to the specialties, can have a large impact on hospital costs, patient outcomes and on the overall efficiency of a hospital. Many papers have analyzed this problem (see for example [1, 2, 3, 4]);

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy
4518264@studenti.unige.it (L. Cademartori); giuseppe.galata@surgiq.com (G. Galatà); 4416766@studenti.unige.it (C. Lo Monaco); marco.maratea@unige.it (M. Maratea); marco.mochi@edu.unige.it (M. Mochi); schouten@kth.se (M. Schouten)
http://www.star.dist.unige.it/~marco/ (M. Maratea); https://marcomochi.github.io/website/ (M. Mochi)
0000-0002-1948-4469 (G. Galatà); 0000-0002-9034-2527 (M. Maratea); 0000-0002-5849-3667 (M. Mochi)

in particular, the introduction of an effective MSS lead to efficiency gains at the operating room department: At Beatrix hospital the annual budget for operating room hours is reduced from 12,848 hours to 9,972 hours (22.4% reduction) while the patients operated increased by 7.7% in 2007 respect to 2006, using the same capacity as at the same time surgery duration decreases by 9.0% [5]. The MSS is often considered as an already available input in many healthcare problem solutions but, due to the different aspects that need to be taken into account for computing a valid schedule, the MSS is an interesting combinatorial problem that deserves its own interest. Going in some more details, the MSS problem is the task of assigning the specialties to the available operating rooms in the different days and sessions, taking into account that not all the specialties need to be assigned the same amount of time and that, during the considered days, the amount of time each specialty should be assigned can vary. The aim of the MSS is to support the hospital to organize the resources and plan the different specialties in the next weeks/months. In particular, by developing a MSS early a hospital can properly manage the personnel and the resources, thus leading to a reduction of the costs. Moreover, by helping the hospital to manage the surgeries and reducing the surgery waiting list, a proper solution to the MSS problem is vital to improve the degree of patients' satisfaction. Complex combinatorial problems, possibly involving optimizations, such as the MSS problem, are usually the target applications of AI languages such as Answer Set Programming (ASP). Indeed ASP, thanks to its readability and the availability of efficient solvers, e.g., CLINGO [6], has been successfully employed for solving hard combinatorial problems in several research areas, and it has been also employed to solve many scheduling problems [7, 8, 9, 10, 11], also in industrial contexts (see, e.g., [12, 13, 14] for detailed descriptions of ASP applications).

In this paper we present a mathematical formulation of the MSS problem. We then apply ASP to solve the MSS problem, by presenting a compact ASP encoding obtained by modularly representing input specifications in ASP, and then running an experimental analysis on randomly generated MSS benchmarks, obtained by varying the number of days and trying different scenarios, created with realistic sizes and parameters inspired from data seen in literature. Results using the state-of-the-art ASP solver CLINGO show that ASP is a suitable solving methodology also for the MSS problem, since we are able to solve optimally instances of the MSS problem in few seconds even considering planning horizon up to 180 days. We also compare the performance of our ASP solution to those of top performing Max-SAT, Pseudo-Boolean and ILP solvers run on instances obtained by automated translation of ASP encoding and instances: Results show that CLINGO, run on the ASP encoding contribution of this paper and employing an optimization algorithm based on unsatisfiable cores, is almost always the best option. Finally, we describe the implementation of a web application we have developed in order to support users in the usage of our solution. The application allows for inserting the main parameters of the problem, running CLINGO on the encoding without actually installing nothing locally, and showing results graphically.

The paper is structured as follows. Sections 2 and 3 present an informal description of the MSS problem, and its precise mathematical formulation, respectively. Then, Section 4 shows our ASP encoding, whose experimental evaluation is presented in Section 5. Section 6 describes the implementation of our web application. The paper ends by discussing related work and conclusions in Section 7 and 8, respectively.

2. Problem Description

With the computation of an MSS, a hospital can see in which days, sessions and operating rooms (ORs) each specialty will do the surgeries. This is important since by looking at the MSS the hospital can manage the personnel and the resources in advance. The MSS is thus often scheduled for long periods of time and as soon as possible, to be able to assign the surgery to the patients in time and to properly organize the personnel. To schedule the MSS a hospital should evaluate the percentage of time that needs to be assigned to each specialty and the allowed errors for such a period of time, in order to better respond to the patients' needs. The percentage of assignments is evaluated as the number of times each specialty is assigned divided by the total number of sessions available in the period considered. To produce a proper schedule, the solution must assign the specialties taking into account the percentage targets and the allowed errors of each specialty. At most n sessions are associated to each day, where n is equal to the maximum number of sessions that could be assigned to an OR. Each session is identified by an id. For example, in a hospital with the maximum number of sessions equal to 2, day 1 will be linked to sessions 1 and 2, while day 2 will be linked to sessions 3 and 4, and so on for all the remaining days. Each session is then linked to all the ORs and the scheduler must assign a specialty to each session. Since the MSS is planned for a long period of time, hospitals could desire that the target assignment of each specialty is respected not for all the considered days, but may vary, e.g., on a monthly or weekly basis. Another aspect that could change during the considered period and between the ORs are the sessions. The usage of each OR is often splitted in two sessions for each day but, sometimes, some ORs can be split in a different number of sessions, higher or used even for just one session. In particular, the single-session solution could be used when a specialty requires particular resources and the time to prepare them is long enough that changing the specialty at mid day would be a waste of time. Moreover, some ORs could be unavailable in some days and the scheduler must be able to consider these unavailability.

Overall, the MSS problem takes as input the number of ORs and specialties, the number of days to consider for the scheduling, the number of sessions for each day, and the different target values for each specialty, and computes the assignment of the different specialties to the available ORs of a hospital in the considered planning horizon. An optimal solution minimizes the difference between the percentage of usage of each specialty and the target value of each period. An example of MSS is presented in Table 1. In particular, the table is the result obtained by our solution, that we will show later in the paper, considering 90 days and fixed target value for each month. Moreover, we considered a hospital with 10 ORs, each splitted in 2 sessions in each day, and 5 specialties (these numbers corresponding to hospitals of small-medium size in Italy) SP1 $\ldots$ SP5 : The table shows the MSS for the first 7 days of the solution. In particular, each row represents a day and the sessions linked to that day, the columns report the ORs, and the intersection shows the specialty assigned to the OR in that day and session.

3. Mathematical formulation of the MSS problem

In this section, we provide a mathematical formulation of the basic version of the problem (called Scenario A later).

Table 1
Example of MSS generated by our solution.

Day	Session	OR1	OR2	OR3	OR4	OR5	OR6	OR7	OR8	OR9	OR10
1	1	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
	2	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
2	3	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
	4	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
3	5	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
	6	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
4	7	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
	8	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
5	9	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP3	SP1	SP4
	10	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP2	SP1	SP4
6	11	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP2	SP1	SP4
	12	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP2	SP1	SP4
7	13	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP2	SP1	SP4
	14	SP5	SP5	SP3	SP3	SP2	SP5	SP4	SP2	SP1	SP4

Definition 1. *Let*

- *day be a constant that is equal to the number of days considered;*
- *max_session be a constant that is equal to the maximum number of session associated to an operating room in a day;*
- *s_count be a constant that is equal to day $\times$ max_session and represents the number of sessions that must be assigned to each operating room;*
- $D = \{t : t \in [1..day]\}$ *be the set of all days;*
- $DD = \{(d_1, d_2)_1, \dots, (d_1, d_2)_n\}$ *be a set of n pair of days such that for every pair d_2 is greater than d_1;*
- $OR = \{o_1, \dots, o_m\}$ *be a set of m operating rooms;*
- $SP = \{sp_1, \dots, sp_k\}$ *be a set of k specialties;*
- $S = \{s_1, \dots, s_{s\_count}\}$ *be a set of s_count sessions id;*
- $\delta : OR \times SP \mapsto \{0, 1\}$ *be a function associating an operating room to a specialty such that $\delta(o, sp) = 1$ if the operating room o can be assigned to the specialty sp, and* 0 *otherwise;*
- $\rho : OR \times D \mapsto S$ *be a function associating an operating room and a day to a session id such that $\rho(o_n, d_m) \geq max\_session * d_m$ - (max_session-1) and $\rho(o_n, d_m) \leq max\_session * d_m$;*
- $\varepsilon : SP \times D \times D \mapsto \mathbb{N}$ *be a function associating a specialty, a starting day and an ending day to a value representing the percentage target to reach from the starting day to the ending day;*
- $\omega : SP \times D \times D \mapsto \mathbb{N}$ *be a function associating a specialty, a starting day and an ending day to a value representing the maximum error that is allowed from the starting day to the ending day;*
- $\zeta : SP \times D \times D \mapsto \mathbb{N}$ *be a function associating a specialty, a starting day and an ending day to a value representing the percentage of times that a session has been assigned to the specialty.*

Let $mss : OR \times S \times SP\ D \mapsto \{0,1\}$ *be a function such that* $mss(o,s,sp,d) = 1$ *if the session* s *in the day* d *and in the operating room* o *is assigned to the specialty* sp*, and* 0 *otherwise. Moreover, for a given mss, let* $A_{mss} = \{(o,s,sp,d) : o \in OR, s \in S, sp \in SP, d \in D, mss(o,s,sp,d) = 1\}$.

Then, given sets OR, SP, S, D, DD and functions δ, ρ, ε, ω, ζ, *the MSS problem is defined as the problem of finding a schedule x, such that*

(c_1*)* $|\{\rho(o,d) = s\}|$=1 $\;\forall o \in OR, \forall d \in D, \forall s \in S$;

(c_2*)* $|\{sp : mss(o,s,sp,d) = 1\}|$=1 $\;\forall o \in OR, \forall s \in S, \forall sp \in SP, \forall d \in D, \rho(o,d) = s$;

(c_3*)* $|\{mss(o,s,sp,d) = 1\}|$=0 $\;\forall o \in OR, \forall s \in S, \forall sp \in SP, \forall d \in D, \rho(o,d) = s, \delta(or,sp) = 0$;

(c_4*)* $|\{mss(o,s,sp,d) = 1\}|$=0 $\;\forall o \in OR, \forall s \in S, \forall sp \in SP, \forall d \in D, \rho(o,d) \neq s$;

(c_5*)* $\zeta(sp,d_1,d_2) > 0 \;\; \forall sp \in O, \forall (d_1,d_2) \in DD$;

(c_6*)* $|\varepsilon(sp,d_1,d_2) - \zeta(sp,d_1,d_2)| \leq \omega(sp,d_1,d_2) \;\; \forall sp \in O, \forall (d_1,d_2) \in DD$;

Condition (c_1) ensures that at each operating room is assigned to a session s_{count} times. Condition (c_2) ensures that each operating room, in each day and session is assigned to exactly one specialty. Condition (c_3) ensures that no operating room is assigned to a not allowed specialty. Condition (c_4) ensures that each specialty is assigned to an operating room in the right session and day. Condition (c_5) ensures that the percentage of times a specialty is assigned is bigger than 0 in every range of days required. Condition (c_6) ensures that the percentage target of time a specialty is assigned minus the actual percentage is less than the allowed error.

Definition 2 (Distance target percentage). *Given a solution mss,*
let $t_{mss} = \sum_{sp \in SP, (d_1,d_2) \in DD;} \mid \varepsilon(sp,d_1,d_2) - \zeta(sp,d_1,d_2) \mid$ *. Intuitively,* t_{mss} *represents the sum of the distance between the target percentage and the actual percentage of times each specialty is assigned to the operating rooms in the range between* d_1 *and* d_2*.*

Definition 3 (Optimal solution). *A solution mss is said to dominate a solution mss' if* $|t_{mss}| < |t_{mss'}|$*. A solution is optimal if it is not dominated by any other solution.*

4. ASP Encoding for the MSS problem

We assume the reader is familiar with syntax and semantics of ASP. Starting from the specifications in the previous section, here we present our compact and efficient ASP solution for the MSS problem, organized in two paragraphs containing input and output data model, and the ASP encoding, respectively. The ASP encoding is based on the input language of CLINGO [15]. For details about syntax and semantics of ASP programs we refer the reader to [16].

```
1 session(SID,DAY,OR) :- operatingRoom(OR,_), sessionN(OR,N,DAY), SID=1..s_count, SID >=
      ((max_session*DAY)-(max_session-1)), SID<=((max_session*DAY)-(max_session-N)), not
      inactive(OR,DAY).
2 n_session(N,START,END) :- N = #count{SID,OR,DAY : session(SID,OR,DAY), DAY >= START, DAY <
      END}, targetShare(_,_,_START,END).
3 {mss(OR,SID,SP,DAY) : operatingRoom(OR, SP)} == 1 :- session(SID,DAY,OR).
4 effectiveShare(SP,PERCENTAGE,START,END) :- SESSION = #count{ OR,SID,DAY : mss(OR,SID,SP,DAY), D
      >= START, D < END}, n_session(N,START,END), specialty(SP), PERCENTAGE = ((SESSION*100) /
      N).
5 :- effectiveShare(SP,PERCENTAGE,START,END), targetShare(SP,TARGET,ERROR,START,END), PERCENTAGE
      < (TARGET-ERROR).
6 :- effectiveShare(SP,PERCENTAGE,START,END), targetShare(SP,TARGET,ERROR,START,END), PERCENTAGE
      > (TARGET+ERROR).
7 :- effectiveShare(SP,PERCENTAGE,START,END), PERCENTAGE <= 0.
8 :~ effectiveShare(SP,ES,START,END), targetShare(SP,TS,ERR,START,END). [|ES-TS|@1,SP,START]
```

Figure 1: ASP encoding of the MSS problem

Data Model. The input data is specified by means of the following atoms:

- Instances of `sessionN(OR,N,DAY)` represent the number of sessions (`N`) in which the operating room identified by an id (`OR`) is split in the day (`DAY`).
- Instances of `operatingRoom(OR,SP)` represent which specialty (`SP`) can be assigned to the operating room identified by an id (`OR`).
- Instances of `specialty(SP)` represent the different specialties identified by their id (`SP`).
- Instances of `targetShare(SP,TARGET,ERROR,START,END)` represent for each specialty (`SP`) the target percentage (`TARGET`) of utilization and the maximum distance allowed to the target value (`ERROR`) in the range of days between `START` and `END`.
- Instances of `day(DAY)` represent the available days.

The output is an assignment represented by an atom of the form `mss(OR,SID,SP,DAY)`, where the intuitive meaning is that the operating room with id `OR` in the session with id `SID` and in the day `DAY` is assigned the specialty `SP`.

Encoding. The related encoding is shown in Figure 1, and is described next. To simplify the description, we denote as r_i the rule appearing at line i of Figure 1.

Auxiliary atoms in the heads of rules r_1, r_2 and, r_4 are derived by the encoder to simplify the other rules. In particular, rule r_1 assigns the correct session ids to each operating room for all the days considered. The assignment is made assigning an id such that the number of ids assigned in each active day is equal to the number of sessions in which the operating room is splitted. Rule r_2 evaluates the total number of sessions available in the range of days between the values start and end. This value is then used to evaluate the percentage of assignment of each specialty. Rule r_3 assigns one of the possible specialties to a session of every operating room. Rule r_4 derives an atom that represents the assignment percentage of each specialty. In particular, it counts the number of sessions linked to each specialty and divides it by the total number of sessions that are available in that period. Then, rules r_5 and r_6 check that the percentage of each specialty is compatible with the target values and the allowed errors. Rule r_7 ensures that the percentage of each specialty is bigger than 0. Finally, weak constraint r_8 minimizes the difference between the assigned and target percentage of each specialty in each period of time.

5. Experimental Results

In this section, we report the results of an empirical analysis of the MSS problem via ASP (second paragraph). For the problem, data have been randomly generated using parameters inspired by literature and real world data (first paragraph). A third paragraph compares results obtained with alternative logic-based formalisms. The experiments were run on a AMD Ryzen 5 2600 CPU @ 3.40GHz with 16 GB of physical RAM. The ASP system used was CLINGO [15] 5.4.0, using parameters *--opt-strategy=usc* for faster optimization and *--parallel-mode 4* for parallel execution. This setting is the result of a preliminary analysis (but presented later in Table 3) done also with other parameters, i.e., the default configuration and the one having *--restart-on-model* for optimization. The time limit was set to 30 seconds. Encodings and benchmarks employed in this section can be found at: http://www.star.dist.unige.it/~marco/RuleMLRR2022/material.zip .

MSS benchmarks. Data are based on the sizes and parameters of a typical middle sized hospital, with 5 different specialties and 10 ORs. Each specialty is associated with a target value for each month and an error, that is equal to 10 for all the specialities. Each specialty can be assigned to just some randomly selected ORs and the target value is assigned by dividing the number of ORs in which the specialty can be assigned to the total number of ORs, and adding to the result a random value in the range between -5 and 5. To test our solution we considered four different scenarios. In the first scenario, that we will call Scenario A, we considered to have the constant *max_session* equal to 2, while the constant *d_count* has values from 30 to 180. Moreover, in this scenario the target value for each specialty is equal for each month. For this scenario, we considered 10 instances, each with different target values for all the specialties, for each range of days considered. In particular, we tested the scalability of the scheduler by considering an increasing number of days: 30, 60, 90, 120, 150 and, 180.

Then, we generated a second scenario, that we will call Scenario B, that is based on the Scenario A considering 90 days. The difference with Scenario A is that for each month the target value is increased or decreased by a random value between -2 and 2, thus for each specialty there are three different target values. Changes in the target values could be done by the hospital manager because of different availability of doctors or due to the increase of the surgeries of some specialty.

For the third and fourth scenario, named Scenario C and D, respectively, we again considered a planning horizon fixed to 90 days. The constant *max_session* is equal to 2 for the Scenario C, while for the Scenario D is equal to 3. This means that, in the fourth scenario, one randomly selected operating room is splitted in three sessions. The difference between the Scenario C and the others is that, for 5 days, three ORs are unavailable, meaning that no session can be assigned to them during that days. The scenarios C and D aim thus at evaluating what is the impact of limiting the usage of the ORs, or changing the number of sessions, respectively.

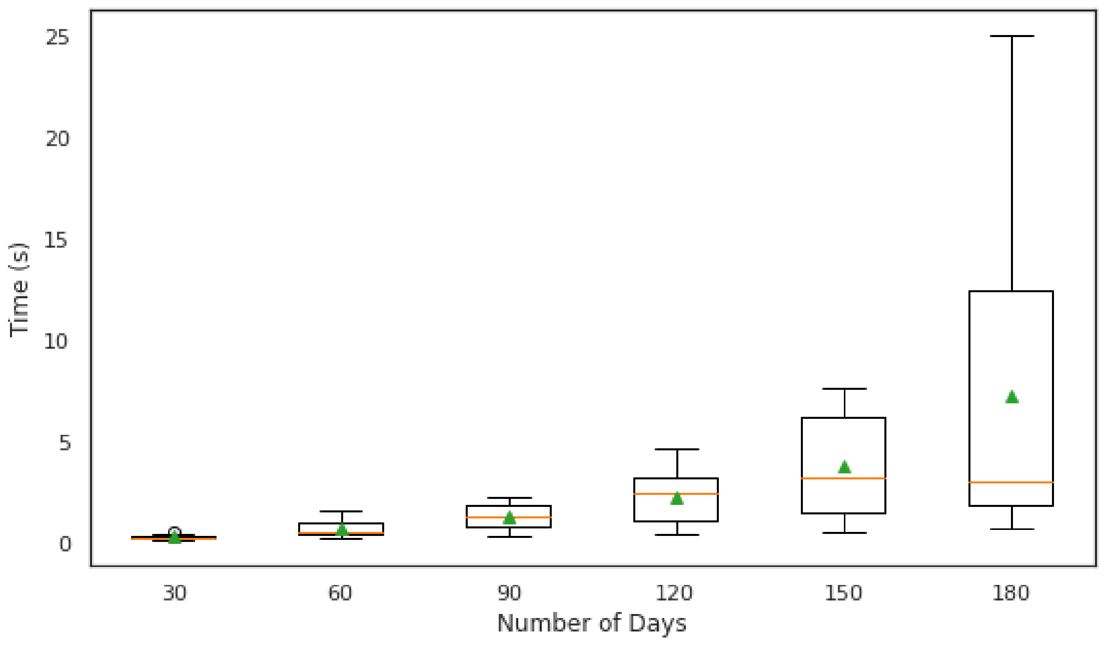


Figure 2: Results obtained by solving 10 instances per group of days in Scenario A. The box starts from the first quartile and ends at the third quartile. The mean time is represented by the (green) triangle, while the (orange) line represents the median value.

Results of our MSS solution. First, we tested the performances of the scheduler in the basic scenario (Scenario A). The results for this scenario are shown in Figure 2, which represents the range of seconds required to reach the optimal solution in all the 10 instances tested with the different number of days considered, identified by the minimum and maximum times for solving the instances in the set, together with the mean and the median time. From the figure it can be seen that the scheduler is able to optimally schedule the MSS in a mean time of less than 10 seconds even considering 180 days of planning horizon, which is a remarkable result. Moreover, besides being able to reach an optimal solution in less than 10 seconds on average, from the figure it can be noted that even in the worst case, the scheduler is able to find the optimal solution in less than 30 seconds.

Then, we tested the performance of the scheduler in the Scenario B. Testing the scheduler with the 10 instances with 90 days in this scenario we found that the scheduler was able to reach the optimal solution on average in 3 seconds, that is a time that is very near to the time required in Scenario A. Thus, this analysis reveals that even changing the target values in each month for all the specialties, our solution maintains very good performance.

Having evaluated now the performance in Scenario A and B, we then tested the scheduler in Scenario C and D. Table 2 reports the time required by each instance in Scenario A and in these more constrained scenarios, on 90 days planning horizon.

From the table we can see that the timing obtained by Scenario C is almost equal to the original one. So, even if three ORs are unavailable for 5 days, the scheduler is able to compute the optimal solution in the same time required in the Scenario A. In the Scenario D, the scheduler obtained

Table 2
Time required for each instances in the different Scenarios and considering 90 days

Instance #	Time (s) Scenario A	Time (s) Scenario C	Time (s) Scenario D
1	1.7	1.7	1.7
2	0.3	0.2	0.3
3	1.8	1.1	4.3
4	2.0	3.0	2.0
5	0.9	2.2	0.9
6	1.9	1.0	2.1
7	0.8	0.6	0.6
8	2.2	1.6	2.3
9	0.9	0.8	0.9
10	0.9	5.2	0.8
Mean	1.3	1.7	1.5

the optimal solution almost in the same time as in the Scenario A for all but one instance: Indeed, the third instance requires 4 seconds instead of 2 seconds to reach the optimal solution (from a preliminary analysis, this harder instance corresponds to a setting in which a higher number of sessions is set to an OR assigned to only one specialty with low target).

Overall, we can say that also in Scenario C and D the scheduler is able to reach highly satisfying results, also when compared to the basic Scenario A.

Comparison to alternative logic-based formalisms. In the following, we present an empirical comparison of our ASP-based solution with alternative logic-based approaches, obtained by applying automatic translations of ASP instances. In more detail, we used the ASP solver WASP [17], with the option `-pre=wbo`, which converts ground ASP instances into pseudo-Boolean instances in the wbo format [18]. Then, we used the tool PYPBLIB [19] to encode wbo instances as MaxSAT instances.

Then, we considered three state-of-the-art MaxSAT solvers, namely MAXHS [20], OPEN-WBO [21], and RC2 [22], and the industrial tool for solving optimization problems GUROBI [23], which is able to process instances in the wbo format. Concerning CLINGO, we used *(i)* its default configuration (CLINGO-DEF); *(ii)* the option `restart-on-model` (CLINGO-ROM); and *(iii)* the option `-opt-strategy=usc` (CLINGO-USC). The latter enables the usage of algorithm OLL [24], which is the same algorithm employed by the MaxSAT solver RC2.

The experiments were executed on Scenario A considering the 10 instances with 30 days horizon, with a timeout of 30 seconds. Results are reported in Table 3, where for each solver and instance we report the ranking obtained by each solver, counting optimal solutions. The solver is in the first position if it finds the solution in the shortest time; a dash means that the solver did not compute the solution before the time limit. As a general observation, CLINGO-USC obtains the best performance overall, since it is the first to find the optimal solution in all but one instance. The performance of CLINGO-ROM is in general slightly worse than the one of CLINGO-USC, even if in the majority of the instances the required time to reach the optimal solution is similar to the time required by CLINGO-USC. GUROBI is able to reach the optimal solutions before

Table 3
Comparison of ASP solution with alternative logic-based solutions.

Instance	CLINGO-DEF	CLINGO-ROM	CLINGO-USC	MAXHS	OPEN-WBO	RC2	GUROBI
1	4	3	2	6	5	-	1
2	4	2	1	-	-	-	3
3	4	2	1	-	-	-	3
4	4	2	1	-	-	-	3
5	4	2	1	-	-	-	3
6	4	2	1	-	-	-	3
7	4	2	1	-	-	-	3
8	4	2	1	-	-	-	3
9	4	2	1	-	-	-	3
10	2	2	1	-	-	-	3

CLINGO-USC and CLINGO-ROM in the first instance while in all the other instances it ranked third. Concerning MaxSAT solvers, we observe that both OPEN-WBO and MAXHS are able to reach the optimal solution before the time limit just in the first instance, while in all the other instances they can not obtain the optimal solution before the time limit. RC2 can not return an optimal solution in any of the instances evaluated. Concerning CLINGO-DEF, we obtain the optimal solution in all the instances but, without using any of the available options, the solutions are obtained in more time than the other options and GUROBI, but for instance 10.

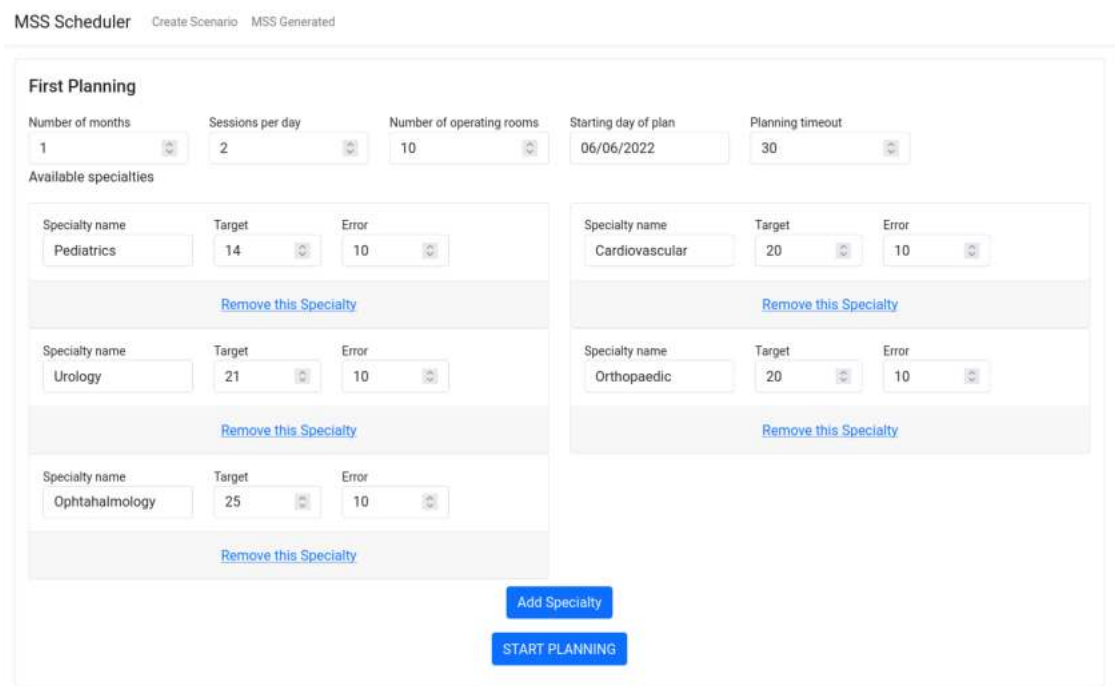
6. Web Application

After having presented our solution and compared it with other solvers, we have wrapped the encoder and the CLINGO solver inside a NodeJS architecture and developed a simple graphical user interface (GUI) to configure the different inputs of the problem. By developing the web app, we want to reduce the burden related to the installation and the proper usage of the ASP-based solution, mainly for non-technical users. Moreover, even if our solution was able to solve all the tested instances in less than 30 seconds, without a proper interface even the best solution could be discarded because of the difficulties caused by the technology itself.

In particular, the first page of the web app, shown in Figure 3, is devoted to the definition of the characteristics of the problem. It allows setting:

- The number of months to schedule.
- The number of sessions per each ORs.
- The number of ORs.
- The starting day of the MSS.
- The different specialties, each with a specific target and error.
- The timeout of the scheduler.

Once the user is satisfied with the inserted data, by clicking on the "START PLANNING" button, she can start the scheduling. The web app processes the data to transform them to a format that allows the compatibility with the CLINGO solver and, once the solver finds the optimal


MSS Scheduler

First Planning

Number of months	Sessions per day	Number of operating rooms	Starting day of plan	Planning timeout
1	2	10	06/06/2022	30

Available specialties

Specialty name	Target	Error
Pediatrics	14	10
Cardiovascular	20	10
Urology	21	10
Orthopaedic	20	10
Ophtahalmology	25	10

Figure 3: The first page of the web app. From this page the user can define the inputs of the problem.

solution or the timeout time is reached, the user is redirected to the second page. In the second page, reported in Figure 4, there are two cards available. In the first card, called "Table", is shown a properly processed result obtained by the solver (assuming that at least a solution is provided, even if not optimal; however, this is not the case of all our analysis). The card shows the MSS obtained by the solver in a table format. In each row of the table there are the day and the linked session plus the information regarding the specialties assigned to the different ORs, thus mimicking the MSS output of Table 1. In the second card, called "Graphics", are shown the graphs comparing the target and the actual percentage of time each specialty is assigned for every range of days considered. This card helps the user to evaluate the quality of the result in a simple way.

7. Related Work

The section is organized in two paragraphs: the first presents works that highlights the importance of solving the MSS problem and alternative methods for solving the problem, while the second mentions works in which ASP has been already successfully employed to closely related scheduling problems.

Solving the MSS problem. In [4] is presented a literature review on how different Operations Research techniques can be applied to the surgical planning. Presenting the different approaches to the MSS problem, the authors pointed out that a more efficient MSS can improve the usage

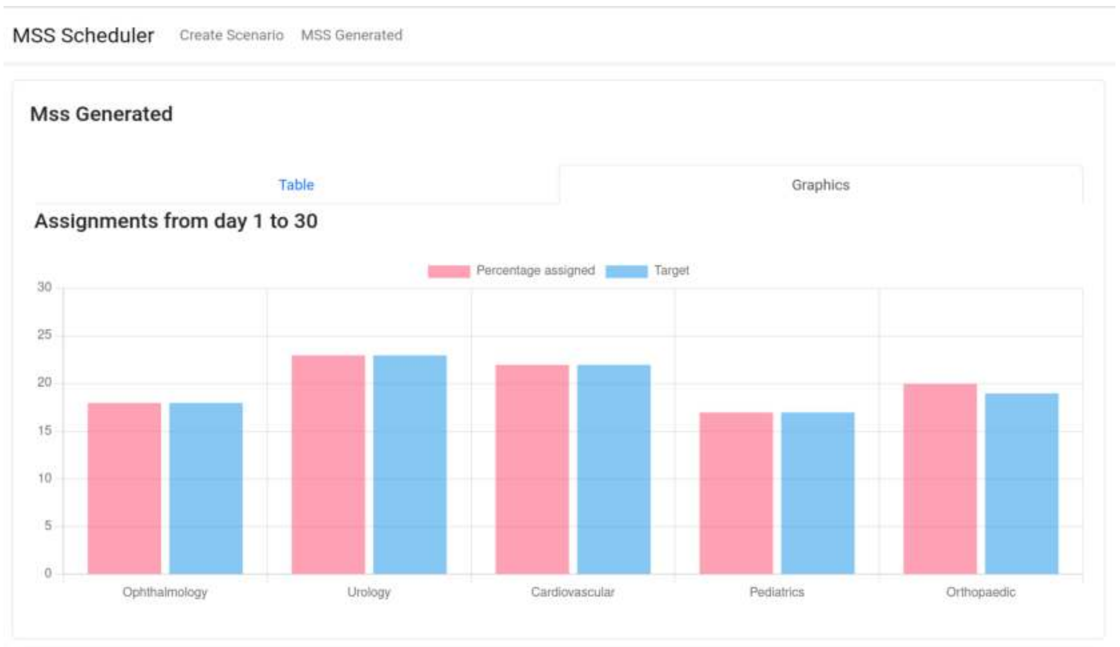


Figure 4: The second page of the web app. From this page the user can evaluate the quality of the MSS obtained by the solver.

of the different resources involved (such as wards, that we do not take into account). [5] shows the benefit of implementing an effective MSS in a regional hospital in the Netherlands. In particular, thanks to the suggestion of the solution proposed, the hospital was able to reduce the budget while increasing the number of patients operated. In this work, the MSS is evaluated as a cyclic schedule composed of different individual surgical case types. Thus, the MSS is composed by a sequence of surgeries instead of blocks of specialties. Moreover, the MSS is planned for 3 weeks only. In [3], the authors proposed a solution to the MSS problem and the surgical case assignments problem formulating it using a mixed integer nonlinear programming approach. They compared their solutions to the historical data of an Australian public hospital. Differently from our work, the solution proposed by the authors maximizes the number of patients operated instead of focusing on target values required by the hospital. The work in [25] used a simulation-optimization approach to solve the MSS problem. In particular, they used a two-stage stochastic optimization model and a discrete-event simulation model to handle uncertainty such as the surgery duration. Differently from our work, they did not consider a target value for the different specialities. The authors of [26] used a mixed-integer linear programming model to address the problem. They used the required surgeries of the week to assign the ORs to the different specialties and considered a fixed (two) number of sessions for each day. In [27], the authors addressed the MSS problem by proposing a cyclic schedule for the frequently performed surgical procedures, maximizing the operating room utilization.

Solving scheduling problems with ASP. ASP has been successfully used for solving hard combinatorial and application scheduling problems in several research areas. In the healthcare domain, the first solved problem was the *Nurse Scheduling Problem* [28, 29, 10], where the goal is to create a scheduling for nurses working in hospital units. Then, the problem of assigning ORs to patients, denoted as *Operating Room Scheduling*, has been treated, and further extended to include bed management [9]. More recent problems include the *Chemotherepy Treatment Scheduling* problem [30], in which patients are assigned a chair or a bed for their treatments, and the *Rehabilitation Scheduling Problem* [11], which assigns patients to operators in rehabilitation sessions. Often problems in which an MSS needs to be computed, including those dealing with the Operating Room Scheduling problem mentioned above, consider the MSS as an input of the problem; however, as we have seen in this paper and by the presence of a number of works at the state-of-the-art dealing uniquely with the problem, the MSS is per se of interest and deserves devoted solutions, to be possibly integrated with other problem solutions building on it.

Concerning scheduling problems beyond the healthcare domain, ASP encoding were proposed for the following problems: *Incremental Scheduling Problem* [31], where the goal is to assign jobs to devices such that their executions do not overlap one another; *Team Building Problem* [7], where the goal is to allocate the available personnel of a seaport for serving the incoming ships; the work in [32], where, in the context of routing driverless transport vehicles, the setup problem of routes such that a collection of transport tasks is accomplished in case of multiple vehicles sharing the same operation area is solved via ASP, in the context of car assembly at Mercedes-Benz Ludwigsfelde GmbH, and the recent survey paper by Falkner et al. [13], where industrial applications dealt with ASP are presented, including those involving scheduling problems.

8. Conclusion

In this paper, we have presented an analysis of the MSS problem modeled and solved with ASP. We started from an informal description of the problem, formulated it in precise mathematical terms, and then presented our ASP solution. Results on synthetic benchmarks show that the ASP solution is able to optimally solve the MSS problem even when considering large planning horizons, up to 6 months. Moreover, solving more difficult scenarios, in which, e.g., targets and number of sessions change within the planning horizon, reduce just slightly the performance of the scheduler. We also compared our solution to other logic-based languages and tools for solving combinatorial problems, on instances obtained by automatic transformation of ASP instances: The analysis shows that, on instances of our basic scenario, our solution with CLINGO employing optimization algorithms based on unsatisfiable cores [33] has the best performance. For what concerns future works, we are currently working on extending our experiments. Moreover, we would like to implement and test optimization algorithms (see, e.g., [34]), and to investigate re-scheduling solutions, that may come into play when the MSS scheduling can not be implemented for some reasons, e.g., sudden unavailability of ORs. Finally, we plan to propose our benchmarks to future ASP Competitions [35].

References

[1] C. Van Riet, E. Demeulemeester, Trade-offs in operating room planning for electives and emergencies: A review, Operations Research for Health Care 7 (2015) 52–69. doi:https://doi.org/10.1016/j.orhc.2015.05.005, proc. of ORAHS 2014.

[2] Y. B. Ferrand, M. J. Magazine, U. S. Rao, Managing operating room efficiency and responsiveness for emergency and elective surgeries—a literature survey, IIE Transactions on Healthcare Systems Engineering 4 (2014) 49–64. arXiv:https://doi.org/10.1080/19488300.2014.881440.

[3] B. Spratt, E. Kozan, Waiting list management through master surgical schedules: A case study, Operations Research for Health Care 10 (2016) 49–64. URL: https://www.sciencedirect.com/science/article/pii/S2211692316300042. doi:https://doi.org/10.1016/j.orhc.2016.07.002.

[4] G. Francesca, R. Guido, Operational research in the management of the operating theatre: a survey., Health care management science 14,1 (2001) 89–114. doi:doi:10.1007/s10729-010-9143-6.

[5] van Oostrum, Jeroen, Applying Mathematical Models to Surgical Patient Planning, Ph.D. thesis, E, 2009. URL: http://hdl.handle.net/1765/16728.

[6] M. Gebser, B. Kaufmann, T. Schaub, Conflict-driven answer set solving: From theory to practice, Artificial Intelligence 187 (2012) 52–89.

[7] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, N. Leone, Team-building with answer set programming in the Gioia-Tauro seaport, Theory and Practice of Logic Programming 12 (2012) 361–381.

[8] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti, P. Wanko, Train scheduling with hybrid ASP, in: LPNMR, volume 11481 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 3–17.

[9] C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, I. Porro, An ASP-based solution for operating room scheduling with beds management, in: P. Fodor, M. Montali, D. Calvanese, D. Roman (Eds.), Proceedings of the Third International Joint Conference on Rules and Reasoning (RuleML+RR 2019), volume 11784 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 67–81.

[10] M. Alviano, C. Dodaro, M. Maratea, Nurse (re)scheduling via answer set programming, Intelligenza Artificiale 12 (2018) 109–124.

[11] M. Cardellini, P. D. Nardi, C. Dodaro, G. Galatà, A. Giardini, M. Maratea, I. Porro, A two-phase ASP encoding for solving rehabilitation scheduling, in: S. Moschoyiannis, R. Peñaloza, J. Vanthienen, A. Soylu, D. Roman (Eds.), Proceedings of the 5th International Joint Conference on Rules and Reasoning (RuleML+RR 2021), volume 12851 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 111–125.

[12] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Magazine 37 (2016) 53–68.

[13] A. A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, E. C. Teppan, Industrial applications of answer set programming, Künstliche Intelligenz 32 (2018) 165–176.

[14] P. Schüller, Answer set programming in linguistics, Künstliche Intelligence 32 (2018) 151–155.

[15] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: ICLP (Technical Communications), volume 52 of *OASICS*, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 2:1–2:15.

[16] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, ASP-Core-2 input language format, Theory and Practice of Logic Programming 20 (2020) 294–309.

[17] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, F. Ricca, Evaluation of disjunctive programs in WASP, in: LPNMR 2019, volume 11481 of *LNCS*, Springer, 2019, pp. 241–255.

[18] Olivier Roussel and Vasco Manquinho, Input/Output Format and Solver Requirements for the Competitions of Pseudo-Boolean Solvers, 2012.

[19] C. Ansótegui, T. Pacheco, J. Pon, Pypblib, 2019. URL: https://pypi.org/project/pypblib/.

[20] P. Saikko, J. Berg, M. Järvisalo, LMHS: A SAT-IP hybrid maxsat solver, in: SAT 2016, volume 9710 of *LNCS*, Springer, 2016, pp. 539–546. URL: https://doi.org/10.1007/978-3-319-40970-2_34. doi:10.1007/978-3-319-40970-2_34.

[21] R. Martins, V. M. Manquinho, I. Lynce, Open-wbo: A modular maxsat solver,, in: SAT 2014, volume 8561 of *LNCS*, Springer, 2014, pp. 438–445. URL: https://doi.org/10.1007/978-3-319-09284-3_33. doi:10.1007/978-3-319-09284-3_33.

[22] A. Ignatiev, A. Morgado, J. Marques-Silva, RC2: an efficient maxsat solver, J. Satisf. Boolean Model. Comput. 11 (2019) 53–64. URL: https://doi.org/10.3233/SAT190116.

[23] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual, 2021. URL: https://www.gurobi.com.

[24] A. Morgado, C. Dodaro, J. Marques-Silva, Core-Guided MaxSAT with Soft Cardinality Constraints, in: CP 2014, Springer, Lyon, France, 2014, pp. 564–573.

[25] T. R. Bovim, M. Christiansen, A. N. Gullhav, T. M. Range, L. Hellemo, Stochastic master surgery scheduling, European Journal of Operational Research 285 (2020) 695–711.

[26] I. Marques, M. E. Captivo, N. Barros, Optimizing the master surgery schedule in a private hospital, Operations Research for Health Care 20 (2019) 11–24. URL: https://www.sciencedirect.com/science/article/pii/S2211692318300225.

[27] J. van Oostrum, M. van Houdenhoven, J. Hurink, E. Hans, G. Wullink, G. Kazemier, A master surgical scheduling approach for cyclic scheduling in operating room departments, OR Spectrum = OR Spektrum 30 (2008) 355–374. doi:10.1007/s00291-006-0068-x.

[28] C. Dodaro, M. Maratea, Nurse scheduling via answer set programming, in: LPNMR, volume 10377 of *LNCS*, Springer, 2017, pp. 301–307.

[29] M. Alviano, C. Dodaro, M. Maratea, An advanced answer set programming encoding for nurse scheduling, in: AI*IA, volume 10640 of *LNCS*, Springer, 2017, pp. 468–482.

[30] C. Dodaro, G. Galatà, A. Grioni, M. Maratea, M. Mochi, I. Porro, An ASP-based solution to the chemotherapy treatment scheduling problem, Theory and Practice of Logic Programming 21 (2021) 835–851.

[31] M. Balduccini, Industrial-size scheduling with ASP+CP, in: J. P. Delgrande, W. Faber (Eds.), Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings, volume 6645 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 284–296.

[32] M. Gebser, P. Obermeier, T. Schaub, M. Ratsch-Heitmann, M. Runge, Routing driverless

transport vehicles in car assembly with answer set programming, Theory and Practice of Logic Programming 18 (2018) 520–534.

[33] M. Alviano, C. Dodaro, Unsatisfiable core analysis and aggregates for optimum stable model search, Fundam. Informaticae 176 (2020) 271–297. URL: https://doi.org/10.3233/FI-2020-1974. doi:10.3233/FI-2020-1974.

[34] E. DiRosa, E. Giunchiglia, M. Maratea, A new approach for solving satisfiability problems with qualitative preferences, in: M. Ghallab, C. D. Spyropoulos, N. Fakotakis, N. M. Avouris (Eds.), ECAI, volume 178 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2008, pp. 510–514.

[35] M. Gebser, M. Maratea, F. Ricca, The seventh answer set programming competition: Design and results, Theory and Practice of Logic Programming 20 (2020) 176–204.