A Mechanism for Reasoning over Defeasible Preferences in Arg2P ⋆ ⋆⋆

Giuseppe Pisano[1 [0000−0003−0230−8212]], Roberta Calegari[1 [0000−0003−3794−2942]], Andrea Omicini[2 [0000−0002−6655−3869]], and Giovanni Sartor[1 [0000−0003−2210−0398]]

[1] ALMA–AI Interdepartmental Center of Human Centered AI
[2] Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum–Università di Bologna, Italy
g.pisano@unibo.it, roberta.calegari@unibo.it, andrea.omicini@unibo.it, giovanni.sartor@unibo.it

Abstract. This paper introduces argumentation over defeasible preferences in Arg2P, an argumentation framework based on logic programming. A computational mechanism is first implemented in Arg2P according to Dung's defeasible preference model, then generalised to enable arbitrary preference relations over arguments.

Keywords: Argumentation · Defeasible preferences · Arg2P

1 Introduction

This work focuses on argumentation with conditional preferences within the Arg2P (short for Arg-tuProlog) framework [10, 2]. The topic is already tackled from a theoretical perspective by various works – e.g., [11, 7, 8, 6] – but, to the best of our knowledge, a complete working implementation of this functionality does not exist, yet [1]. Dung *et al.* [6] cope with the problem of defeasible preferences while maintaining compatibility with the standard approach to the semantics of standard abstract argumentation [5]—thus providing the basis for the implementation. Accordingly, in this paper we discuss first of all an efficient implementation of the model presented in [6] along with some examples.

Then, an improvement of the computational mechanism for dealing with defeasible preferences is proposed in order to support a broader set of argument ordering relations—like the ones introduced in ASPIC$^+$ weakest/last and elitist/democrat. The choice of ordering is very important in the construction of an argumentation framework, and it heavily depends on the application area. A higher number of supported orderings implies then a huge advantage in terms

⋆ G. Pisano, R. Calegari and G. Sartor have been supported by the H2020 ERC Project "CompuLaw" (G.A. 833647). A. Omicini has been supported by the H2020 Project "StairwAI" (G.A. 101017142).

of applicability of the tool. Since the original model [6] only supports a smaller set of orderings, and does not deal with aggregated preferences, the final goal of our generalisation is to overcome this limitation.

The paper is structured as follows. Section 2 overviews background notions—i.e., the formal argumentation framework, the conditional preferences model, and the Arg2P language. Section 3 introduces the Arg2P implementation of the argumentation framework [6] along with some examples. Section 4 discusses the improvement of the previously implemented mechanisms enabling the use of the arguments orderings introduced in ASPIC+ [9]. Section 5 concludes the work.

2 Preliminary notions

In the following we introduce the structured argumentation model adopted in Arg2P (Subsection 2.1)—an ASPIC$^+$-like argumentation system [5, 9]. Preferences are introduced in Subsection 2.2 according to the model presented in [6]. Finally, the Arg2P language is discussed in Subsection 2.3.

2.1 Structured Argumentation Framework (AF)

In this section we introduce the argumentation language, arguments and attack relations between them. As usual a literal is an atomic proposition or its negation.

Notation 1 *For any literal ϕ, its complement is denoted by $\bar{\phi}$. That is, if ϕ is a proposition p, then $\bar{\phi} = \neg p$, while if ϕ is $\neg p$, then $\bar{\phi}$ is p.*

Literals are brought into relation through rules.

Definition 1 (Rules). *A **strict rule** r has the form: $\rho : \quad \phi_1, ..., \phi_n \rightarrow \psi$ while a **defeasible rule** has the form: $\rho : \quad \phi_1, ..., \phi_n \Rightarrow \psi$ in both cases with $0 \leq n$, and where ρ is the unique identifier for r, denoted by $\mathsf{N}(r)$; each $\phi_1, \ldots, \phi_n, \psi$ is a literal; the set $\{\phi_1, \ldots, \phi_n\}$ is denoted by Antecedent(r) and ψ by Consequent(r).*

A strict rule is an expression indicating that if $\phi_1, \ldots, \phi_n$ hold, then without exception it holds that ψ. Strict rules are used for representing strict (sound) knowledge and are denoted with StrictRules. Defeasible rules – denoted with DefRules – are rules that can be defeated by contrary evidence. Pragmatically, a defeasible rule is used to represent defeasible knowledge, i.e., tentative information that may be used if nothing could be posed against it. For simplicity, we define axioms and non-axiom premises via strict and defeasible rules with empty *Antecedent*. A theory consists of a set of rules.

Definition 2 (Theory). *A **defeasible theory** is a tuple $\langle \mathsf{Rules} \rangle$ where $\mathsf{Rules} \subseteq \mathsf{StrictRules} \cup \mathsf{DefRules}$.*

Arguments are built from strict and defeasible rules. Given the rules of a defeasible theory, we can construct arguments by chaining them, as specified in the following definition, cf. [9].

Definition 3 (Argument). *An* ***argument*** A *constructed from a defeasible theory* $\langle \mathsf{Rules} \rangle$ *is a finite construct of the form:* $A : A_1, \ldots, A_n \rightarrow_r \phi$ *or* $A : A_1, \ldots, A_n \Rightarrow_r \phi$ *with* $0 \leq n$*, where*

- r *is the top rule of* A*, denoted by* $\mathsf{TopRule}(A)$*;*
- A *is the argument's unique identifier;*
- $A_1, \ldots, A_n$ *are* A*'s* direct subarguments*, denoted by* $\mathsf{DirectSub}(A)$*;*
- $\mathsf{Sub}(A)$ *denotes the entire set of subarguments of* A*, i.e.,* $\mathsf{Sub}(A) = \mathsf{Sub}(A_1) \cup \ldots \cup \mathsf{Sub}(A_n) \cup \{A\}$*;*
- ϕ *is the* conclusion *of the argument, denoted by* $\mathsf{Conc}(A)$*;*
- $\mathsf{DefRules}(A)$ *is the set of defeasible rules exploited to build argument* A
- $\mathsf{LastDefRules}(A)$ *is the set of the last defeasible rules exploited to build argument* A *where*

$$\mathsf{LastDefRules}(A) = \begin{cases} \mathsf{TopRule}(A) & \text{if } \mathsf{TopRule}(A) \text{ is defeasible} \\ \bigcup_{A_n \in \mathsf{DirectSub}(A)} \mathsf{LastDefRules}(A_n) & \text{otherwise} \end{cases}$$

Arguments can be in conflict, accordingly to two kinds of attack: rebuts and undercutting, defined as in [9]. We assume the postulates of consistency and closure for argument-based systems ensuring that strict rules cannot entail contradictory conclusions—i.e., we assume that *1)* the set of strict rules is closed under transposition, and *2)* the strict closure of the empty set is directly consistent [4].

Definition 4 (Attack). *An argument* A ***attacks*** *an argument* B *at* $B' \in \mathsf{Sub}(B)$ *iff* A *undercuts or rebuts* B *(at* B'*), where: (i)* A *undercuts* B *(at* B'*) iff* $\mathsf{Conc}(A) = \neg \mathsf{N}(r)$ *and* B'*s top rule* r *is defeasible; (ii)* A *rebuts* B *(at* B'*) iff* $\mathsf{Conc}(A) = \neg\phi$*,* $\mathsf{Conc}(B') = \phi$ *and* B'*'s top rule* r *is defeasible. We say that an argument* A directly rebuts *(*dbut*) or* directly undercuts *(*dcut*)* B *if* A *rebuts or undercuts* B *at* B *itself.*

An argumentation graph can then be defined exploiting arguments and attacks.

Definition 5 (Argumentation framework and graph). *An* ***argumentation framework****, aka also as abstract argumentation framework, constructed from a defeasible theory* T *is a tuple* $\langle \mathcal{A}, \rightsquigarrow \rangle$*, where* $\mathcal{A}$ *is the set of all arguments constructed from* T*, and* $\rightsquigarrow$ *is the* attack *relation over* $\mathcal{A}$*. The corresponding* ***argumentation graph*** *is the corresponding directed graph where: arcs are interpreted as attacks and nodes are arguments.*

The expression **structured argumentation framework** in the following specifies that the internal construction of the arguments is provided in the framework (in particular as rules and their chaining).

Notation 2 *Given an argumentation graph* $G = \langle \mathcal{A}, \rightsquigarrow \rangle$*, we write* $\mathcal{A}_G$*, and* $\rightsquigarrow_G$ *to denote the graph's arguments and attacks respectively. We also write* $\mathsf{dbut}_{\rightsquigarrow}$ *and* $\mathsf{dcut}_{\rightsquigarrow}$ *to denote the set of direct rebuts and direct undercuts.*

2.2 Preference-Based Argumentation Framework

Dung *et al.* [6] introduce a conservative generalisation of structured argumentation frameworks aimed at dealing with arguments expressing preferences over rules. Based on that, in the following we define a preference-based argumentation framework. First, we define preference arguments and the corresponding set.

Definition 6 (Preference Argument). *A **preference argument** constructed from a defeasible theory T is an argument A such that $\mathsf{Conc}(A)$ has the form $N(r1) \prec N(r2)$ where $r1$ and $r2$ are in $\mathsf{DefRules}$.*

Notation 3 (Preference Arguments Set) *We denote the set of preference arguments in an argumentation framework with $\mathcal{A}_P$.*

Given the notion of preference arguments, we introduce the notion of *Preference-Based AF* (PAF) so as to reason over such arguments [6]. In short, a PAF is a transformation of a standard AF also accounting for preference arguments. This means to define how preference arguments interact with other arguments—i.e., how admissibility of one argument $A \in \mathcal{A}_P$ impacts on others arguments, and more precisely, how the attack relation between these new arguments is defined.

The key idea in [6] is to transform the AF direct attacks into arguments that can then be attacked and challenged by preference arguments. Then, conditional preferences attack these new arguments, thus challenging their effect. Converting attacks into arguments leads to the need of rebuilding the attacks set since new arguments have to be considered—namely, *(a)* attacks coming from attacks transformed into arguments, and *(b)* attacks involving preference arguments.

Following this idea, in PAF the attack relation is built taking into account *a)* and *b)*. With respect to *a)*, existing AF direct attacks – i.e., $\mathsf{dbut}, \mathsf{dcut}$ – are transformed into arguments and added to the argument set $\mathcal{A}$. All the attacks belonging to the $\rightsquigarrow$ set are transformed consistently.

Notation 4 *Let us denote with* datt *the set of all the attacks from transformation "attacks into arguments". Note that attacks in* datt *are attacks from old attacks transformed into arguments to either an original argument already existing in AF or to an old attack also transformed into an argument.*

Example 1. For instance, let us consider an argumentation framework AF, with three arguments – A, B and B_1 where $\mathsf{Sub}(B_1) = \{B\}$ – and two attacks: one direct rebut from A to B, and one rebut attack from A to B_1. The transformation (point *a)*) only impacts the first attack (being a direct attack). A new argument ATT representing the attack from A to B is so added to the new argumentation framework (PAF), and consequently, also the corresponding attack from ATT to B. As concern the attack from A to B_1, it is not necessary to convert it into an argument – not being a direct attack – but it is enough to change the source of the B_1 attack from A to ATT. The resulting argumentation framework PAF will be so composed of four arguments – namely A, B, B_1, ATT – and two attacks—one from ATT to B and one from ATT to B_1 (see Fig. 1).

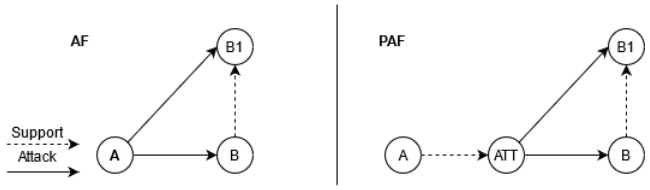

Fig. 1. Standard argumentation graph (left) and transformed preference graph attack relation w.r.t. *a)* (right).

With respect to *b)*, new attacks are introduced, capturing the contrast between a preference argument and an attack argument—i.e., reifying the notion of a preference. The idea is that the preference for argument A over argument B is an attack against the view that B can successfully attack A: prevailed-over arguments cannot attack the prevailing ones. It follows that if a preference argument according to which A prevails over B is admissible, then the attack from B into A is rejected from the extension containing the preference argument.

Notation 5 *Let us denote with* patt *the set of all attacks from a single preference argument to an attack argument.*

Example 2. Let us consider the PAF from Example 1. Consider then a fourth argument P claiming the superiority of argument B over argument A—this is for the sake of simplicity since arguments actually express preferences over defeasible rules. With respect to *b)*, this scenario would result in a new attack from P to ATT, where ATT is the argument representing the attack from A to B (Fig. 2). In fact, being B preferred to A, it would be impossible for the latter to attack the former.

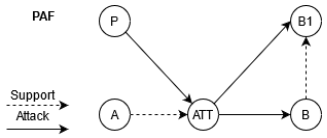

Fig. 2. Example 2 transformed preference graph attack relation w.r.t. *a)*.

Formally, a PAF is defined as (definition slightly modified w.r.t. the original one to match the general model introduced in Subsection 2.1):

Definition 7 (Preference-Based AF). *Given an argumentation framework* $T = \langle \mathcal{A}, \rightsquigarrow \rangle$*, a* ***preference-based argumentation framework*** *of* T *is an abstract argumentation framework* $FK = \langle \mathcal{A}_T, \rightsquigarrow_T \rangle$ *where*

- $\mathcal{A}_T = \mathcal{A} \cup \mathit{dbut}_{\rightsquigarrow} \cup \mathit{dcut}_{\rightsquigarrow}$*, and*

– $\rightsquigarrow_T = datt \cup patt$ *where:*
  - $datt \subseteq (\mathsf{dbut}_{\rightsquigarrow} \cup \mathsf{dcut}_{\rightsquigarrow}) \times \mathcal{A}_T$ *such that* $((A, B), X) \in datt$ *iff*
    * $X \in \mathcal{A}$ *and* $B \in \mathsf{Sub}(X)$*, or*
    * $X = (C, D) \in (\mathsf{dbut}_{\rightsquigarrow} \cup \mathsf{dcut}_{\rightsquigarrow})$ *and* $B \in \mathsf{Sub}(C)$
  - $patt \subseteq \mathcal{A} \times \mathsf{dbut}_{\rightsquigarrow}$ *such that* $(X, (A, B)) \in patt$ *iff* $\exists d \in \mathsf{LastDefRules}(A)$ *such that* $d < \mathsf{TopRule}(B) \in \mathsf{Conc}(X)$.

The key point in the original work by [6] is the analysis of the relation used to identify the attacks in *patt*. This analysis is outside the scope of this paper being our analysis more focused on the computational properties of the AF to PAF transformation. Hence, for further details we point to the original work. Note that our definition of *patt* includes a specific use of preferences to determine (in) effective attacks, i.e., a rebutting attack fails if directed against a preferred rule in the attacked argument.

2.3 Arg2P syntax

Arg2P [10] is a modular framework for structured argumentation based on logic programming [3]. In this section we briefly introduce the Arg2P language necessary for understanding the examples discussed in the paper. For a more detailed introduction of the tool please refer to [10].

In Arg2P, rules can be expressed by exploiting the operators $\Longrightarrow = \{->, =>\}$ – where $->$ stands for *strict* implication while $=>$ for the defeasible one – with the notation: $label : premise_1, \ldots, premise_n \implies conclusion.$ where *label* is the identifier (unique name) of a rule and $premise_1, \ldots, premise_n$ are the premises that entail the *conclusion*.

Accordingly, axioms/ defeasible axioms take the form: $label : [] \Longrightarrow conclusion.$ Premises and conclusions can take any form accepted in Prolog: i.e., all Prolog terms – e.g. atoms, variables, lists, compound terms – are allowed. The binary attack relation between arguments – which underpins rebutting attacks – can be reached via terms' negation. Strong negation – negation as definite falsity – is encoded prepending the $-$ operator to a term (i.e. $-term$). Undercut attacks can be expressed exploiting the notation: $label_2 : premise_1, \ldots, premise_n \implies undercut(label_1)$ where $label_1$ is the identifier of a defeasible rule in the theory. Preferences over rules can be expressed with the notation $sup(r1, r2)$ where $r1$ and $r2$ are the rules unique identifiers. The Arg2P framework implements a *graph-based* mode providing as output the entire argumentation graph according to the specified semantics: the evaluation can be required via the `buildLabelSets` predicate.

3 Defeasible preferences mechanism in Arg2P

The model presented in Subsection 2.2 is well-suited to be implemented in Arg2P: being it self-contained, it can be easily modularised and integrated with the existing model. The transformation from AF to PAF fully generalises mechanisms

and semantics used for standard argumentation frameworks, thus enabling the reuse of standard tools already implemented in the framework. Conceptually, we can handle the PAF model as a function F that, given an argumentation framework A_F, returns a new argumentation framework B_F. How we evaluate and handle framework B_F w.r.t. A_F is left unchanged. Accordingly, the implementation of the model in Arg2P only requires the creation of a new optional module responsible for the framework's transformation. Users can then determine how to handle preferences, by combining the most suitable modules for the evaluation. The remainder of this section shows how function F is defined and implemented, according to the model in Subsection 2.2.

3.1 Transformation algorithm

Listing 1.1. Transformation algorithm

```
1  (PafArguments, PafAttacks) BuildPafArgumentationGraph(Arguments, Attacks):
2          % Filters attacks that are in no circumstances impacted by preference arguments
3          (ValidAttacks, InvalidAttacks) = filterSupRelatedAttacks(Attacks)
4          % Transformation point a) converts the preference-related attacks into arguments,
5          % also transposing the connected attacks, datt set
6          (NewArguments, NewAttacks) = convertAttacks(InvalidAttacks)
7          % Transformation point b) build the patt set
8          PrefAttacks = buildPrefAttacks(Arguments, NewArguments)
9          % Returns the new arguments and attacks sets
10         PafArguments = append(Arguments, NewArguments)
11         PafAttacks = append(ValidAttacks, NewAttacks, PrefAttacks)
12         return (PafArguments, PafAttacks)
```

Listing 1.1[3] shows the pseudo-code of the algorithm used in transforming AF to PAF—i.e., the encoding of the function F.

First of all, attacks are split into two sets: non influent attacks for the purpose of transformation – i.e., attacks not affected by preferences, that can be translated as-is in the final framework `ValidAttacks` – and attacks that should be considered as invalid and thus transformed— `InvalidAttacks` (line 3 Listing 1.1). After that, the transformation of the framework begins. First, direct attacks are converted into arguments also adding the new derived attacks—point *a)* of the transformation (see Subsection 2.2) (line 6 Listing 1.1). Then, conforming to the transformation point *b)* (see Subsection 2.2) attacks from preference arguments to the newly created attack arguments are built, always in accordance to the preference attack relation—line 8 Listing 1.1.

Note that w.r.t. the original algorithm, the filtering of the attacks (line 3 in Listing 1.1) is introduced in order to enhance the comprehensibility of the final graph and improve the computational performance. The set $(dbut_{\rightsquigarrow} \cup dcut_{\rightsquigarrow})$ is filtered to exclude from the transformation those attacks not referenced by an element in *patt*; then the transformation is applied only to the `InvaildAttacks` subset, while the remaing `ValidAttacks` are transposed as-is in the final graph. From a computational perspective, the graph is kept as small as possible and most faithful to the original, thus avoiding useless transformations.

[3] Sources and technical details are available on the official project page http://arg2p.apice.unibo.it

Proposition 1. *The transformation algorithm in 1.1 is sound w.r.t. the model in Subsection 2.2 by [6].*

Proof. The transformation algorithm is almost completely faithful to the original model, then assuring the soundness of its results w.r.t. the original. The only difference is in the filtering part, where we avoid transforming those attacks not influenced by any preference. It is easy to prove the introduced optimisation does not impact the general soundness of the transformation algorithm. It is already proved in [6] that a *PAF*, in absence of preference arguments, delivers the same results of a standard *AF*. As a consequence, if an attack does not interact with a preference argument, it is irrelevant if it has been transformed or not. Thus, the filtering of the irrelevant attacks does not impact the final outcome of the evaluation w.r.t. the not filtered graph.

Example 3. Let us consider the Sherlock Holmes example proposed in [6]. A murder investigation involving three persons – namely *p1*, *p2* and *s* – is considered. It is known for sure that the murderer acted alone, *s* was physically unable to kill the victim, and *p1* benefited from the murder.

Listing 1.2. Example 3 rules in Arg2P

```
r1 : inno(p1), inno(s) -> -inno(p2).
r2 : inno(p2), inno(s) -> -inno(p1).
r3 : inno(p1), inno(p2) -> -inno(s).

r4 : beneficiary(p1) -> have_motive(p1).
r5 : beneficiary(p2) -> have_motive(p2).

pi1 : have_motive(p1), -have_motive(p2) -> sup(d2, d1).
pi2 : -have_motive(p1), have_motive(p2) -> sup(d1, d2).

d : [] => inno(s).
d1 : [] => inno(p1).
d2 : [] => inno(p2).
d3 : [] => -have_motive(p1).
d4 : [] => -have_motive(p2).

f1 : [] -> inno(s).
f2 : [] -> beneficiary(p1).
```

Listing 1.2 shows the Arg2P encoding of the case. Strict rules `r1`, `r2`, and `r3` encodes the fact that only one of the suspects can be the murderer—i.e., if two of them are innocent, then the third is guilty. Strict rules `r4` and `r5` derive the motivation to murder from the possible benefits. Five unconditional defeasible rules work as assumptions: `d`, `d1`, and `d2` encode the principle on which persons are innocent unless proven guilty; `d4` and `d5` assume that *p1* and *p2* have both no motive to kill. There are also two facts, `f1` and `f2`, asserting the knowledge on s' innocence (`f1`) and *p1*'s profit from the murder (`f2`). Finally, two additional strict rules – `pi1` and `pi2` – conclude a preference over rules. In particular, if *p1* had a motivation to kill while *p2* had not, then the assumption on *p2*'s innocence is stronger than *p1*'s one (`pi1`). Conversely, if *p1* hadn't a motive to kill while *p2* had, then the favours are on *p1*'s innocence (`pi2`).

Fig. 3 shows the evaluation results under the grounded semantic in a standard argumentation framework – i.e., with no defeasible preferences handling – while Fig. 4 shows the results with the transformation module enabled. As expected, without any preference on the rules the graph remains undecided—it is not possible to decide on arguments concluding *inno(p1)*(*inno(p2)*) and $\neg inno(p1)$

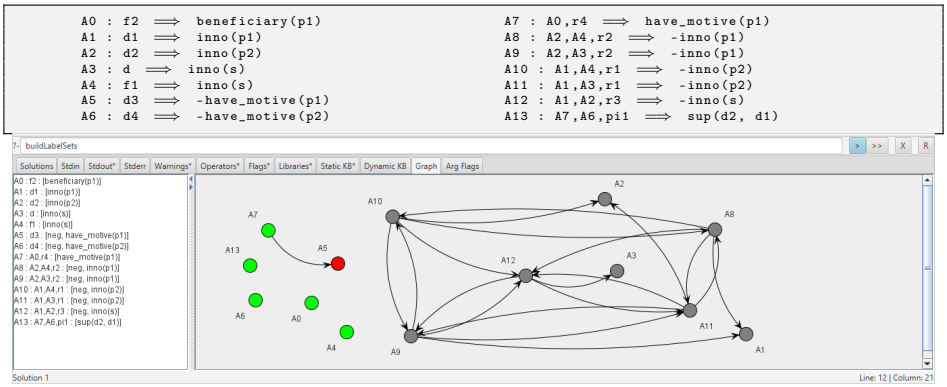

Fig. 3. Arguments from Listing 1.2 with conditional preference disabled.

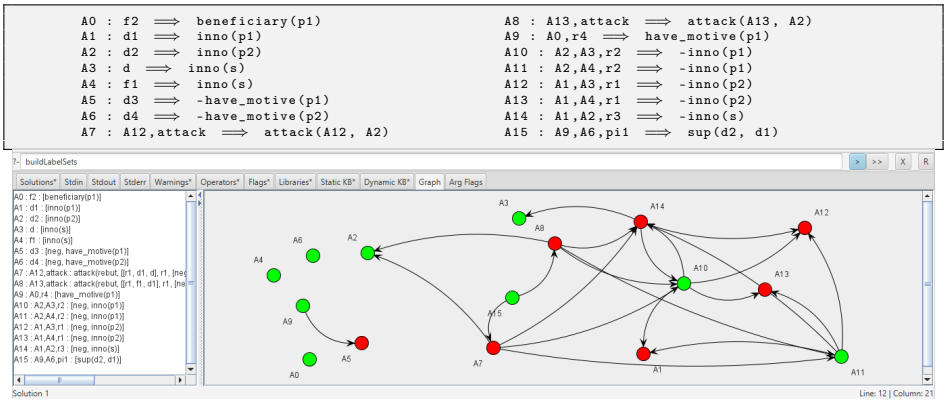

Fig. 4. Arguments from Listing 1.2 with conditional preference enabled.

($\neg inno(p2)$). Conversely, after enabling defeasible reasoning on preferences, it is possible to decide for *p2*'s innocence, thus condemning *p1*.

The latter result is caused by argument *A15* (Fig. 4). The claimed preference of `d2` over `d1` impacts only two direct rebuts: the one from *A14* to *A2* and the one from *A13* to *A2*. Consequently, these are the only attacks transformed into arguments (*A7* and *A8*)—filtering of the transformation algorithm. The attacks coming from these two arguments are the same: they attack argument *A2* and all its derived arguments (*A10*, *A11* and *A14*). If we evaluate the graph, we have that the preference argument (*A15*) is undisputed. It follows the rejection of the two attack arguments *A7* and *A8* and the admissibility of argument *A2* on *p2*'s innocence. The rest of the graph is computed accordingly.

Example 4. We now consider an example of preference reasoning from [11] showing a case of multiple nested preferences. Listing 1.3 shows the Arg2P theory.

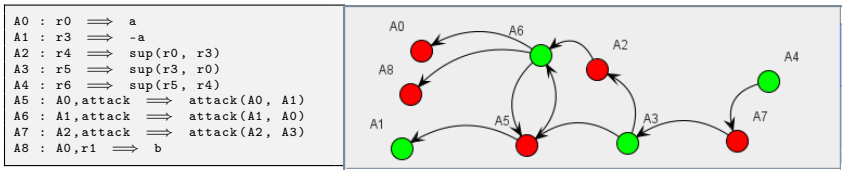

Fig. 5. Arguments from Listing 1.3 with conditional preference enabled.

Listing 1.3. Example from [11] in Arg2P

```
r0 : [] => a.
r1 : a => b.
r3 : [] => -a.
r4 : [] => sup(r0, r3).
r5 : [] => sup(r3, r0).
r6 : [] => sup(r5, r4).
```

Two unconditional rules are stated – `r0` and `r1` – concluding the two conflicting literals a and $\neg a$. Then a defeasible rule (`r1`) claims b if a is proved, and three unconditional rules conclude a preference: `r4` concluding the priority of `r0` over `r3`, the priority of `r3` over `r0` (`r5`) and the priority of `r5` over `r4` (`r6`).

Taking into account no preference, the evaluation under the grounded semantic is undecided: arguments for a and $\neg a$ rebut each other and consequently also the one concluding b is undecided. Intuitively, `r5` states the strength of the $\neg a$ claim and `r6` enforces this priority against the one in `r4`. Accordingly, we expect the argument for $\neg a$ to be accepted thus rejecting the opponent arguments.

Fig. 5 shows the PAF built on the theory and its evaluation under the grounded semantic. Argument $A1$ for $\neg a$ is accepted—thus causing the rejection of arguments $A0$ (a) and $A8$ (b). The accepted preferences are two: the one on `r5` over `r4`, that is undisputed, and the one on `r3` over `r0`. As a consequence, the attack from argument $A2$ to $A3$ (argument $A7$) and the one from $A0$ to $A1$ (argument $A5$) are both rejected.

4 Generalising the reasoning mechanism

The main limitation of the model described in Subsection 2.2 is the impossibility of dealing with attacks coming from a set of preferences, a.k.a. group preference attacks: preference attacks are only allowed from a single preference to an attack. The issue is well known in the literature and already discussed in papers dealing with defeasible preferences, see for instance [8] where an extended argumentation framework handling group attacks is proposed. The problem was also tackled in the work by Modgil [7], where authors leverage a sort of super-arguments to join preferences. This approach was then refined in the logical model proposed in [8].

We believe that the super-argument approach could be an optimal solution to integrate group preference attacks in the Arg2P framework. The extension of the computational mechanism under this perspective requires just a minor modification to the transformation algorithm (listing 1.1), maintains all the computational properties of the standard implementation, and allows overcoming the

limit of the model in Subsection 2.2. Hence, following the insights from [7], we propose a generalisation of the already-presented reasoning mechanism coping with the group preference attack problem and thus enabling the use of different arguments ordering relations. In a nutshell, we leverage artificial arguments joining sets of preferences. Attacks come no more directly from preferences but are issued from these artificial arguments. In the following, we examine in detail our approach with some examples. We cannot here address the corresponding extension of the formal model of Subsection 2.2, which we leave to future works.

4.1 The new algorithm

In the model discussed in Subsection 2.2 attacks $\in$ *patt* can be expressed in the form $(X, (A, B))$, where X is a preference argument and (A, B) is an attack transformed into an argument. In the mechanism generalisation X is no longer a preference argument but rather an artificial argument constructed from the involved preference arguments. In order to cope with such an extension, we first introduce the grouped preference set $\mathcal{A}_{GP}$ defined as:

Definition 8 (Grouped Preference Set). *Given a set of preference arguments $\mathcal{A}_P$ in an argumentation framework, the set of all the possible subsets of $\mathcal{A}_P$, i.e., $\mathcal{A}_{GP} = 2^{\mathcal{A}_P}$ is the **grouped preference set**.+*

Based on grouped preference set, we define a further kind of argument complementing definition 3.

Definition 9 (Joint preferences argument). *Every argument set $A \in \mathcal{A}_{GP}$ is called a **joint preferences argument**. The **direct subarguments** of a joint preferences argument A are all preference arguments $X \in A$.*

We also say that an argument A is preferred to an argument B according to the preferences in a joint preferences argument $X \in \mathcal{A}_{GP}$ with the notation $\succ_X$. Then, we modify definition 7 as follows:

Definition 10 (Generic Preference-Based AF). *Given an argumentation framework $T = \langle \mathcal{A}, \rightsquigarrow \rangle$, a **generic preference-based argumentation framework** (GPAF) of T is an abstract argumentation framework $FK = \langle \mathcal{A}_T, \rightsquigarrow_T \rangle$ where*

- $\mathcal{A}_T = \mathcal{A} \cup \mathsf{dbut}_{\rightsquigarrow} \cup \mathsf{dcut}_{\rightsquigarrow} \cup \mathcal{A}_{GP}$, *and*
- $\rightsquigarrow_T =$ *datt* $\cup$ *patt where:*
  - *datt* $\subseteq (\mathsf{dbut}_{\rightsquigarrow} \cup \mathsf{dcut}_{\rightsquigarrow}) \times \mathcal{A}_T$ *such that* $((A, B), X) \in$ *datt iff*
    - $X \in \mathcal{A}$ *and* $B \in \mathsf{Sub}(X)$, *or*
    - $X = (C, D) \in (\mathsf{dbut}_{\rightsquigarrow} \cup \mathsf{dcut}_{\rightsquigarrow})$ *and* $B \in \mathsf{Sub}(C)$
  - *patt* $\subseteq \mathcal{A}_{GP} \times \mathsf{dbut}_{\rightsquigarrow}$ *such that* $(X, (A, B)) \in$ *patt iff* $B \succ_X A$.

Intuitively, we add to the arguments all the existing preference aggregations in $\mathcal{A}_{GP}$. Then, attacks in *patt* are not determined by single arguments (elements of $\mathcal{A}_P$) but by joint preferences arguments (elements of $\mathcal{A}_{GP}$). If we consider once again Example 2, the corresponding GPAF would contain a new argument AP with P as direct subargument, and the preference attack would come from AP and not from P as in the standard PAF (Fig. 6).

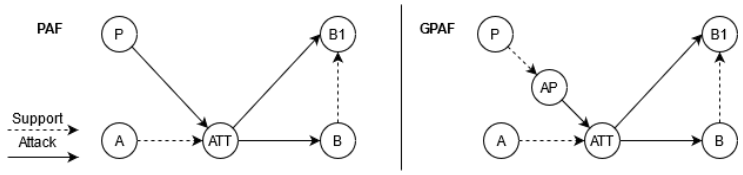

Fig. 6. PAF (left) and transformed GPAF (right).

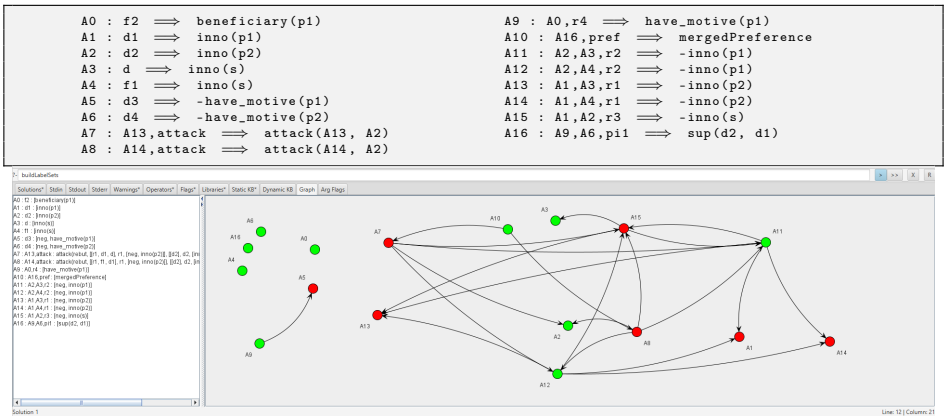

Fig. 7. Evaluation of Example 3 with the extended mechanism and the original PAF's preference attack relation.

Proposition 2. *The generic Preference-Based AF is a conservative generalisation of a PAF when maintaining the PAF preference-attack relation.*

Proof. Let us consider a PAF. For each $A \in patt$ such that $A = (X, (B, C))$ a new argument $\{X\} \in \mathcal{A}_{GP}$ is built, and the *patt* set is modified accordingly—i.e., $(patt - A) \cup \{(\{X\}, (B, C)\}$. We have to prove that the modified graph delivers the same results as the original one. Since $\{X\}$ like any argument in $\mathcal{A}_{GP}$ cannot be directly rebutted/undercut, (being artificially added to the argumentation framework) status of $\{X\}$ in GPAF is the same as that of X in PAF, since X is the only direct subargument of $\{X\}$. It follows that the PAF evaluation remains unaltered w.r.t. the initial transformation.

W.r.t. the algorithm presented in Subsection 3.1, the new transformation involves an improved filtering strategy – taking into account preferences' combination – and an extended construction of preference attacks. Indeed, the construction of the preference attacks should return also the artificial arguments—containing also the attacks derived from their sub-arguments.

Fig. 7 shows the evaluation of Example 3 with the generalised algorithm and the original PAF's preference attack relation. As expected, the result is analogous to Fig. 4, the only difference is the preference attack origin. While in the original

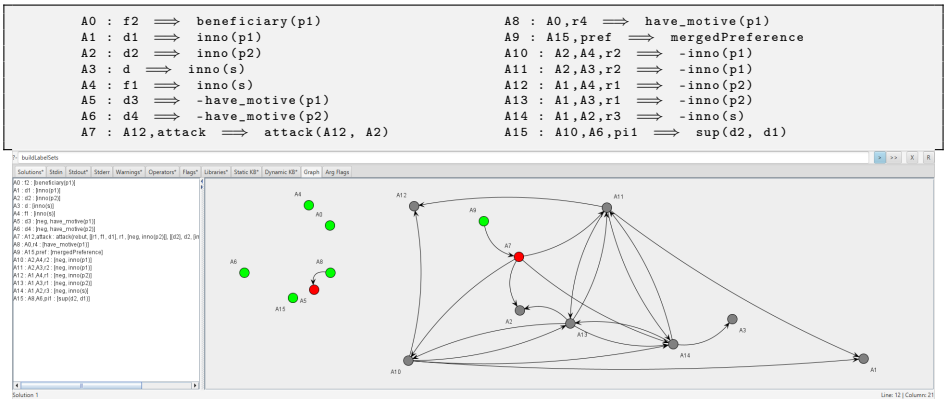

Fig. 8. Example 3 evaluation with the modified algorithm and the weakest democrat argument ordering.

model *A15* directly attacks arguments *A7* and *A8*, in the generalised mechanism (Fig. 7) the artificial argument *A10* is the source of the attack.

From a technical perspective, the improved reasoning mechanism enables different arguments orderings while evaluating the argumentation graph. The evaluation under different orderings could no longer adhere to the assumptions in [6]. However, some application scenarios can consider the loosening of such assumptions, therefore exploiting other types of ordering. For instance, let us consider the orderings introduced in [9]. The authors bear the idea that – abstracting from the specific properties – the choice of an argument ordering is deeply connected to the application scenario. For example, the weakest link principle – according to which all the defeasible steps an argument construction should be considered in the ordering – could be more suited to empirical reasoning scenarios.

Let us consider the application of the weakest-link democratic ordering [9] to Example 3. In a nutshell, in the weakest democrat ordering, an argument A is preferred to an argument B if for every defeasible rule r used to build B, there is a preference of the form $r_1 \succ r$, with r_1 an A's rule. Fig. 8 shows the evaluation results. Using this ordering, preference on `d2` over `d1` is not enough to conclude *p2*'s innocence. In fact, while this preference is enough to conclude the superiority of argument *A2* on *A12*, the same does not hold for argument *A13*, since it based also on the defeasible rule `d`. The example demonstrates the flexibility of the mechanisms presented in this section.

Example 5. Let us now consider an example exploiting joint preference arguments containing a combination of preferences—Listing 1.4. The theory contains two unconditional rules – `r0` and `r2` – concluding the two literals a and $\neg b$. Then a defeasible rule (`r1`) claims b if a, and two unconditional rules conclude a preference: `r3` concluding the priority of `r2` over `r1`, and `r4` concluding the priority of `r2` over `r0`. The GPAF evaluation with the weakest-link democratic ordering and grounded semantic is: accepted for $\neg b$ ($A1$) and rejected for b ($A5$) (Fig. 9).

Listing 1.4. Example 5 theory

```
r0 : [] => a.
r1 : a => b.
r2 : [] => -b.
r3 : [] => sup(r2, r1).
r4 : [] => sup(r2, r0).
```

```
A0 : r0  ⟹  a
A1 : r2  ⟹  -b
A2 : r4  ⟹  sup(r2, r0)
A3 : r3  ⟹  sup(r2, r1)
A4 : A5,attack  ⟹  attack(A5, A1)
A5 : A0,r1  ⟹  b
A6 : A2,A3,pref  ⟹  mergedPreference
```

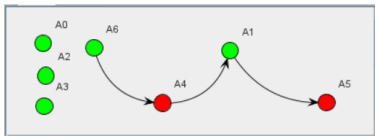

Fig. 9. Arguments from Listing 1.4 with conditional preferences enabled.

$A6$ is the joint preference argument which combines the preferences expressed by $r4$ ($A2$) and $r3$ ($A3$).

5 Conclusions

In this paper we show how the conditional preferences mechanism from [6] can be implemented in Arg2P, then improved in order to cope with arbitrary preference relations over arguments.

The work can be expanded in various ways, starting from a full theoretical foundation of the model. From a pure computational perspective, the work needs further analysis w.r.t. the themes of *(i)* computational complexity and *(ii)* termination—i.e., prove if the algorithms always guarantee a solution or graph configurations leading to undecidable state exist. While have no final result on complexity *(i)*, it should be polynomial w.r.t. the number of input attacks for both the algorithms. Algorithms are based on two main operations: the transformation of the attacks in arguments – linear w.r.t. the number of attacks – and the recognition of the new pref-attacks – requiring in the worst case the examination of all the preference arguments for every attack. So, in the worst case, the complexity should be $O(2N * PA)$, where N is the number of the input attacks and PA is the number of preference arguments. Termination *(ii)* is deeply connected with formal foundation of the model, hence it will be tackled after its completion. Nonetheless, initial informal analysis suggests a transformed graph should be always be obtainable by the proposed algorithms. However, the soundness of solutions provided by the transformed argumentation graph must be proved, in particular w.r.t. corner configurations. In fact, PAF properties hold only when the *defeasible theory* respects some constraints [6]—a.k.a. as preference stratification, or p-stratification. Basically, it is required to ensure partial ordering over rules concluding a preference to avoid cycles in the inference chain—i.e., a preference should not influence the rules it derives from.

We also intend to extend this work towards Arg2P's query-based mode [10]. Intuitively, it would be enough to evaluate also preference arguments that con-

tribute to the *patt* set while evaluating an attacking argument. Further and deeper investigations are required in order to prove its feasibility.

References

1. Calegari, R., Contissa, G., Lagioia, F., Omicini, A., Sartor, G.: Defeasible systems in legal reasoning: A comparative assessment. In: Legal Knowledge and Information Systems. JURIX 2019: The Thirty-second Annual Conference, Frontiers in Artificial Intelligence and Applications, vol. 322, pp. 169–174. IOS Press (11-13 Dec 2019). https://doi.org/10.3233/FAIA190320
2. Calegari, R., Contissa, G., Pisano, G., Sartor, G., Sartor, G.: Arg-tuProlog: a modular logic argumentation tool for PIL. In: Legal Knowledge and Information Systems. JURIX 2020: The Thirty-third Annual Conference. Frontiers in Artificial Intelligence and Applications, vol. 334, pp. 265–268 (9-11 Dec 2020). https://doi.org/10.3233/FAIA200880
3. Calegari, R., Omicini, A., Sartor, G.: Explainable and ethical AI: A perspective on argumentation and logic programming. In: AIxIA 2020 – Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 12414, pp. 19–36. Springer Nature (2021). https://doi.org/10.1007/978-3-030-73065-9_2
4. Caminada, M., Amgoud, L.: On the evaluation of argumentation formalisms. Artificial Intelligence **171**(5-6), 286–310 (2007). https://doi.org/10.1016/j.artint.2007.02.003
5. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77**(2), 321–358 (1995). https://doi.org/10.1016/0004-3702(94)00041-X
6. Dung, P.M., Thang, P.M., Son, T.C.: On structured argumentation with conditional preferences. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019. pp. 2792–2800. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33012792
7. Modgil, S.: Reasoning about preferences in argumentation frameworks. Artificial Intelligence **173**(9-10), 901–934 (2009). https://doi.org/10.1016/j.artint.2009.02.001
8. Modgil, S., Prakken, H.: Reasoning about preferences in structured extended argumentation frameworks. In: Computational Models of Argument: Proceedings of COMMA 2010. Frontiers in Artificial Intelligence and Applications, vol. 216, pp. 347–358. IOS Press (2010). https://doi.org/10.3233/978-1-60750-619-5-347
9. Modgil, S., Prakken, H.: The $ASPIC^+$ framework for structured argumentation: a tutorial. Argument Computation **5**(1), 31–62 (2014). https://doi.org/10.1080/19462166.2013.869766
10. Pisano, G., Calegari, R., Omicini, A., Sartor, G.: Arg-tuProlog: A tuProlog-based argumentation framework. In: Calimeri, F., Perri, S., Zumpano, E. (eds.) CILC 2020 – Italian Conference on Computational Logic. CEUR-WS, vol. 2719, pp. 51–66 (13-15 Oct 2020), http://ceur-ws.org/Vol-2710/paper4.pdf
11. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of Applied Non-Classical Logics **7**(1), 25–75 (1997). https://doi.org/10.1080/11663081.1997.10510900