Modular Logic Argumentation in Arg-tuProlog

Roberta Calegari[1], Giuseppe Contissa[1], Giuseppe Pisano[1(✉)], Galileo Sartor[2], and Giovanni Sartor[1]

[1] Alma AI – Alma Mater Research Institute for Human-Centered Artificial Intelligence, Alma Mater Studiorum—Università di Bologna, Bologna, Italy
{roberta.calegari,giuseppe.contissa,g.pisano,giovanni.sartor}@unibo.it
[2] University of Torino, Turin, Italy
galileo.sartor@unito.it

Abstract. A modular extension of Arg-tuProlog, a light-weight argumentation tool, is here presented and discussed, highlighting how it enables reasoning with rules and interpretations of multiple legal systems. Its effectiveness is demonstrated with examples from different national private international law (PIL) laws, running in Arg-tuProlog. PIL addresses overlaps and conflicts between legal systems by distributing cases between the authorities of such systems (jurisdiction) and establishing what rules these authorities have to apply to each case (choice of law).

Keywords: Modular argumentation · Private international law · Arg-tuProlog

1 Introduction

In our increasingly pervasive and interconnected world, the application and enforcement of the law make it necessary to take into account the interplay of multiple normative systems, especially when dealing with international contracts and other commercial and social interactions involving different countries. Moreover, normative systems may also interact or conflict on different levels: this is true of both national legal systems and of various transnational or international laws and conventions. All these sources of law need to be considered to properly reason about the law.

The research in this paper focuses on the field of private international law (PIL) – a growing and important domain of the law – which deals with the coexistence of multiple normative systems, having distinct and often contradictory rules, and the legal interaction of persons connected to different legal systems,

Roberta Calegari, Giuseppe Pisano and Giovanni Sartor have been supported by the H2020 ERC Project "CompuLaw" (G.A. 833647). Giuseppe Contissa and Galileo Sartor have been supported by the European Union's Justice programme under Grant Agreement No. 800839 for the project "InterLex: Advisory and Training System for Internet-related private International Law".

S. Bandini et al. (Eds.): AIxIA 2021, LNAI 13196, pp. 91–103, 2022.
https://doi.org/10.1007/978-3-031-08421-8_7

trying to establish priorities between them. Conflicts about competences and rules are addressed by identifying which authority is responsible for making a decision in each given case (jurisdiction), and which set of norms should be applied (applicable law).

A recent logical analysis of PIL has highlighted how this body of law can be suitably modelled by modular argumentation [8] so as to provide a formal model of the interaction among multiple legal systems. The model proceeds from the assumption that PIL is not concerned with specific inconsistencies between the rules of different legal systems, since only one system will be selected and applied, regardless of how the others would regulate the same case (choice of law). Thus, the law is modelled through sets of modules in which different legal systems are represented separately. Moreover, each legal-system module is further split into separate modules, each with a specific function: determining jurisdiction, establishing the law to be applied, and providing substantive legal outcomes. This formal model has not yet been captured and implemented in ready-to-use technology.

For this reason, we are here presenting an extension for the Arg-tuProlog framework [5,10] – a lightweight argumentation tool – enabling the exploitation of modular knowledge bases. In this work, previous works on Arg-tuProlog [3] are extended focusing on modularity issues and fully addressing a complete case study in the field of PIL and legal reasoning. Arg-tuProlog makes it possible to design and define knowledge organised in distinct and separate modules that can "call" one another. Such calls request skeptical or credulous reasoning. The final answers from the system are obtained by way of dialectical argumentation. In particular, a knowledge module – which may represent a legal system or parts of it – can be used by itself, or by referring to another module for specific issues. This second approach is done by directly calling and querying the relevant module.

In past years, research in either legal theory or AI and law has devoted little attention to the logical analysis of PIL. Only recently have several projects begun to fill this gap[1], providing computable representations of international and national PIL rules. This is an important development since private international law is an increasingly relevant domain of the law – and considering as well that legal relationships involving citizens of different countries are becoming increasingly frequent. The tool presented in this paper is a further advancement, for it can be used to expand existing projects (such as Interlex), and also as a basis for broader applications. Indeed, the model – and its technology – can also be useful for governing interactions and coordination between heterogeneous agents, belonging to different and differently regulated virtual societies, without recourse to a central regulatory agency [4].

[1] Among these is the European project Interlex, aimed at developing a consultative and training system for internet-related PIL, making it available as an online platform. The platform will be composed of three modules: a Decision Support Module (DSM), a Find Law Module (FLM), and a Training Module (TM). In this context, the core component of the Decision Support Module (DSM) lies in a set of logic representations in Prolog, providing basic legal reasoning capabilities.

Accordingly, the paper is organised as follows. Section 2 presents two examples in the PIL domain illustrating the interaction between national and international legislation. Section 3 then presents the Arg-tuProlog modular argumentation tool. In Sect. 4 the examples discussed in Sect. 2 are represented in the Arg-tuProlog framework. Section 5 discusses the results of the experiment and proposes future lines of research.

2 The Domain of Private International Law: Running Examples

In this section, we will provide two examples of a possible interaction between national and transnational normative systems. In particular, we will focus on one of the EU's main PIL instruments, the Brussels Regulation[2], providing common EU rules on jurisdiction and the recognition and enforcement of judgments. According to the Brussels Regulation, there are some cases where the regulation itself does not give an answer to the question of jurisdiction, pointing instead to national legislation for the relevant laws. This happens, for example, in the sections on consumer contracts (Sect. 4) and third-party proceedings (Sect. 5).

We have built two examples that set aside EU legislation and focus on the switch/conflict between national laws. In our examples, we focus in particular on two sets of national PIL laws: the Italian and the Bulgarian. The source texts presented here are extracted from then English translations of national laws available on the Interlex portal[3].

Example 1 (General jurisdiction rule)
In this example we consider article 3.1 of the Italian Law No. 218 of 31 May 1995 (Reform of the Italian System of Private International Law) and article 4 of the Bulgarian Law DB, bp. 42 ot 17.05.2005 r. (Private International Law Code).

> *Article 3 (Scope of jurisdiction)*
> *1. Italian courts shall have jurisdiction if the defendant is domiciled or resides in Italy or has a representative in this country who is enabled to appear in court pursuant to Article 77 of the Code of Civil Procedure, as well as in the other cases provided for by law. [...]]*

Thus Italian courts shall have jurisdiction if the defendant is domiciled or resides in Italy.

> *Article 4. General Jurisdiction*
> *(1) The Bulgarian courts and other authorities shall have international jurisdiction where: 1. the defendant has a habitual residence, statutory seat or principal place of business in the Republic of Bulgaria; [...]*

[2] Regulation (EU) No. 1215/2012 on jurisdiction and the recognition and enforcement of judgments in civil and commercial matters (recast) (the Brussels Regulation). The EU's two other main PIL instruments are Regulation (EC) No. 593/2008 on the law applicable to contractual obligations (Rome I) and Regulation (EC) No. 864/2007 on the law applicable to noncontractual obligations (Rome II).

[3] https://interlex-portal.eu/FindLaw/.

Thus Bulgarian courts shall have jurisdiction if the defendant has a habitual residence, statutory seat, or the principal place of business in Bulgaria.

Let us consider, as a first scenario, the case of Marius, an Italian citizen with his primary residence in the city of Rome. Marius is summoned to appear in front of a judge to answer a complaint brought against him. Based on this information we can determine that the Italian court of Rome should be assigned jurisdiction in this complaint.

In a second scenario, Marius is also the owner of a business in Bulgaria. In this case, the Bulgarian PIL law – called by the Brussels Regulation – would assign jurisdiction to a Bulgarian court. Since both rules are valid, the jurisdiction in Marius's case belongs to both the Italian and Bulgarian courts. If no priority was set, then a conflict of laws would arise, with two equally valid indications of jurisdiction.

Example 2 (Jurisdiction related to rights in rem)
In the second example we consider articles 5.1 of Italian PIL and article 12 of Bulgarian PIL.

> *Article 5 (Actions concerning rights in rem in immovables situated abroad)*
> *1. Italian courts shall have no jurisdiction over actions concerning rights in rem in immovables situated abroad.*

Thus, under Italian law, Italian Courts have no jurisdiction over actions concerning rights in rem in immovables (real property) situated outside Italy.

> *Article 12. Jurisdiction in Matters Relating to Rights in Rem*
> *(1) (Amended, SG No. 59/2007) The matters under Article 109 of the Code of Civil Procedure relating to immovable property situated in the Republic of Bulgaria, the matters relating to the enforcement or to security which such property constitutes, as well as the matters relating to transfer or establishment of rights in rem in such property, shall be exclusively cognizable in the Bulgarian courts and other authorities. [...]*

Thus, under Bulgarian law, Bulgarian Courts have jurisdiction for matters relating to the transfer or the establishment of rights in rem in immovable property situated in Bulgaria.

Let us consider Marius, an Italian citizen, owner of two houses, one in Italy (Milan) and the other in Bulgaria (Sofia). A claim is brought against Marius with an action concerning a right in rem over one of his immovable properties. Depending on which house is the object of the claim, Marius will be summoned in front of an Italian or Bulgarian judge respectively.

3 Modular Argumentation in Arg-tuProlog

Arg-tuProlog [5,13] is a lightweight modular argumentation tool that fruitfully combines modular logic programming and legal reasoning. It makes it possible to represent, reason, and carry out an argument on conditional norms featuring obligations, prohibitions, and (strong or weak) permissions – including under any burden-of-persuasion constraints that may apply – fully supporting the modular argumentation model, i.e., allowing theory fragmentation, thus enabling the coexistence of different modules.

The approach is based on common constructs in computational argumentation modules – rule-based arguments, argumentation graphs, and labelling semantics – laying their foundation on Dung's abstract argumentation [7] and structured argumentation [2]. Arguments are formed by chaining applications of inference rules into inference trees or graphs – i.e., arguments are constructed using deductive inference rules that license deductive inferences from premises to conclusions (cf. [9]).

The Arg-tuProlog-structured argumentation framework adopts an ASPIC$^+$-like syntax [11]: in a nutshell, arguments are produced from a set of defeasible rules, and attack relationships between arguments are drawn in argumentation graphs. The arguments in the graph are then labelled by applying an acceptance labelling semantics (namely, grounded semantics [1]) that takes burdens of persuasion into account. The framework addresses burdens of persuasion within an argumentation setting [6] (formal accounts of the adopted deontic extensions are discussed in detail in [12], while the implemented burden-of-persuasion model can be found in [6]). The argumentation model is then enhanced according to the concept of modularity [8], making it possible to separate knowledge as well as to create an internal structure of the knowledge (linked modules corresponding to knowledge organisation).

Being completely based on logic programming, the system makes possible a completely integrated cooperation between logic programming and argumentation, therefore – as in the following examples – the knowledge can contain strict rules on which basis to perform queries and reasoning.

3.1 Modular Logic: Architecture and Predicates

In the following, we will focus on deepening the discussion of the modular extension of Arg-tuProlog, being the core of this work and being functional to the appropriate design of private international law. Details on the argument model and its syntax and architecture can be found in [14].

The Arg-tuProlog framework leverages the underlying tuProlog engine and is freely available at [13]. The entire framework is a collection of tuProlog-compatible libraries, and all the required components exploit the tuProlog feature to allow the inclusion of external libraries during the evaluation process. The system's inner modular architecture greatly enhances the upgradability and flexibility of the entire system by making it possible to add new features or modify requirements.

As mentioned, the framework fully supports a modular argumentation model, i.e., it allows theory fragmentation and enables the coexistence and interaction of different modules. Different modules can be combined when querying the system, leading to different responses according to the modules considered. It is also possible to nest modules, thus establishing a hierarchy among the modules. These features make the system particularly suitable for designing and implementing complex scenarios such as the one relating to the PIL domain. Indeed, as discussed in Sect. 2, this legal domain is based on the coordination and cooperation of different legal systems, such as national law and international treaties. From a computational point of view, this mixture translates into a scenario where a logic theory (module) can query or consult a piece of information contained in another theory (module).

Modules are identified by distinct Prolog files (`.pl` files) and can be called and executed exploiting the predicate `module_call(+Modules, :Query)`, where `Modules` is an input parameter containing the list of the required modules – i.e., modules that need to be loaded to answer the query – and `Query` is the query that must be evaluated. In particular, the predicate: *i)* creates a new environment that contains *only* the required modules data, *ii)*, executes the query in the newly created environment, *iii)* and feeds the result to the caller—note that the original caller environment is not altered by the procedure.

The location of the modules must be included in the *root* theory (main module) through the predicate `modulesPath(++FileSystemPath)`, where the input parameter `FileSystemPath` denotes the full path file-system name. Note that in the case of nested calls, it is not necessary to re-specify the path in the submodules.

For example, consider the case in which there is a unique module in the modules folder (`systemPath/modules/moduleOne.pl`) containing the following theory:

```
legislation("italian legislation", moduleOne).
jurisdiction(C) :- legislation(C, ModuleName).
```

In order to use `moduleOne`, the root module needs to set the modules path as shown in the following. In order to better understand the system's functioning, we will add a fact `legislation` to the root module as well.

```
modulesPath(systemPath/modules).
legislation("root legislation", root).
jurisdiction(C):-modules_call([moduleOne], jurisdiction(C)).
```

Then, when calling the goal

```
:- jurisdiction(C)
```

the *module_call*/2 loads the required modules, in this case `moduleOne`; creates the new environment; and proceeds to the query evaluation. The result will be

```
C="italian legislation", ModuleName=moduleOne
```

highlighting that the root theory content is not considered during the query evaluation (i.e., `C="root legislation", ModuleName=root` is not a solution).

If we add a second module – *moduleTwo.pl* – containing the following theory:

```
legislation("bulgarian legislation", moduleTwo).
```

we can combine modules' content – and their solutions – writing the root module as follows:

```
modulesPath(systemPath/modules).
legislation("root legislation", root).
jurisdiction(C)  :- modules_call([moduleOne, moduleTwo],
                          jurisdiction(C)).
```

in such a case, the `legislation(L,X)` goal will provide the two distinct solutions

```
C="italian legislation", ModuleName=moduleOne

C="bulgarian legislation", ModuleName=moduleTwo
```

By exploiting the predicates just discussed, it is so possible to organize the knowledge into a series of distinct modules that may be dependent on and/or pertinent to a hierarchical structure. Accordingly, a dispatch of concerns and contents is possible while ensuring easy interaction and cooperation among distinct modules.

4 Running Examples in Arg-tuProlog

In the following, the examples discussed in Sect. 2 are reified in the Arg-tuProlog framework to show the technology's effectiveness and potential[4].

Both examples discussed in Sect. 2 can be mapped starting from the Brussels Regulation, the Italian national law, and the Bulgarian national law. For such a reason they have been mapped onto the Arg-tuProlog framework exploiting three distinct modules: one for the Brussels Regulation, one for the Italian national law, and one for the Bulgarian national law. In fact, the complexity of the scenario makes it necessary to take different bodies of law into account and to make them interact and interoperate, while also providing a tool for detecting possible conflicts and inconsistencies. For this reason, a modular approach is required.

In the following, we list an extract from the Brussels Regulation codification (*BrusselsRegulation.pl* module) that makes it possible to establish jurisdiction according to the content of the articles. In particular, the final choice on the complaint is referred to the national laws of the UE member states in question. This connection is modelled using the modularity feature of Arg-tuProlog and in particular the *call_module* predicate:

```
hasJurisdiction(Article, Country, Court, ClaimId):-
    personRole(PersonId, ClaimId, defendant),
    memberState(MemberLaw),
    call_module([MemberLaw, ClaimId],
        hasJurisdiction(Article, Country, Court, ClaimId)).
```

[4] The theories used in the examples can be found at https://github.com/tuProlog/arg2p/tree/master/example-theories/IPL-brussels.

```
hasJurisdiction(Article, Country, Court, ClaimId):-
    claimObject(ClaimId, rightsInRem),
    memberState(MemberLaw),
    call_module([MemberLaw, ClaimId],
        hasJurisdiction(Article, Country, Court, ClaimId)).
```

The Italian law module – *italy.pl* – is a simple theory that includes the Prolog translation of the articles from the Italian PIL law as described in Sect. 2. The articles may be represented in the Arg-tuProlog system as follows:

```
hasJurisdiction(art3_1, italy, Court, ClaimId) :-
    personRole(PersonId, ClaimId, defendant),
    personDomicile(PersonId, italy, Court).

hasJurisdiction(art3_1, italy, Court, ClaimId):-
    personRole(PersonId, ClaimId, defendant),
    personAgent(AgentId, PersonId),
    personDomicile(AgentId, italy, Court).

hasJurisdiction(art51, italy, Court, ClaimId):-
    claimObject(ClaimId, rightsInRem),
    immovableProperty(ClaimId, italy, Court).
```

The Bulgarian national law is represented by the *bulgaria.pl* module which contains the Prolog translation from the Bulgarian PIL law as described in Sect. 2. A possible Arg-tuProlog representation is as follows:

```
hasJurisdiction(art4_1, bulgaria, Court, ClaimId):-
    personRole(PersonId, ClaimId, defendant),
    personDomicile(PersonId, bulgaria, Court).

hasJurisdiction(art4_1, bulgaria, Court, ClaimId):-
    personRole(PersonId, ClaimId, defendant),
    personPlaceOfBusiness(PersonId, bulgaria, Court).

hasJurisdiction(art12, bulgaria, Court, ClaimId):-
    claimObject(ClaimId, rightsInRem),
    immovableProperty(ClaimId, bulgaria, Court).
```

Using this knowledge as our basis – the knowledge being split into the corresponding modules – let us discuss the resolution of some example complaints so as to illustrate the potential of using a modular argumentation framework.

Example 1 (General jurisdiction rule). Let us consider the case discussed in Example 2. The facts and details of the case are stored in a separate module (*claim1.pl*), listed in the following. In particular, we have facts establishing the role of Marius (i.e., defendant), his domicile (i.e., Italy), and his place of

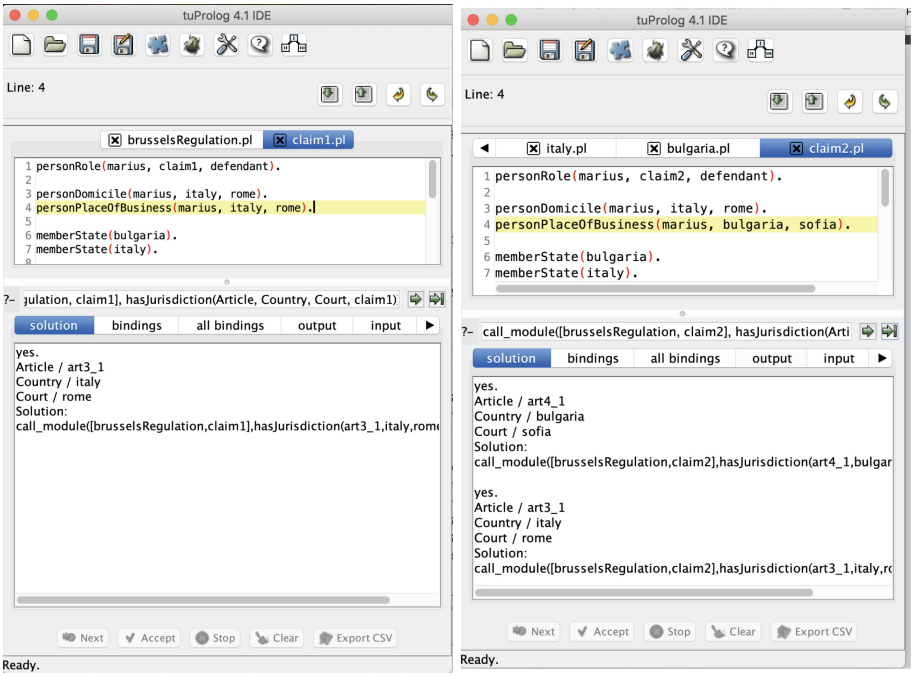

Fig. 1. Arg2p interface: result of claim1 (left) and claim2 (right).

business, which once again is Italy (respectively `personRole`, `personDomicile`, and `personPlaceOfBusiness` predicates). Finally, we have two facts establishing which states are EU member states (under the Brussels Regulation), which in our example are Italy and Bulgaria.

```
personRole(marius, claim1, defendant).

personDomicile(marius, italy, rome).
personPlaceOfBusiness(marius, italy, rome).

memberState(bulgaria).
memberState(italy).
```

To evaluate the case, we can select the jurisdiction simply by calling the following goal over the top module *brusselsRegulation.pl*:

```
call_module([brusselsRegulation, claim1],
    hasJurisdiction(Article, Country, Court, claim1)).
```

Figure 1 (left) shows the result, which is that under article 3.1 of the Italian law, the court to which the case is assigned is in Rome, Italy. The result is

perfectly consistent since the defendant is domiciled in Italy (and also his place of business).

Let us now consider the same case, with the only difference that the place of the defendant's business is in Sofia, Bulgaria (*claim2.pl*).

```
personRole(marius, claim2, defendant).

personDomicile(marius, italy, rome).
personPlaceOfBusiness(marius,bulgaria,sofia).

memberState(bulgaria).
memberState(italy).
```

As shown in Fig. 1 (right), the answer in this case is twofold. Article 4.1 of the Bulgarian law and Article 3.1 of the Italian law should apply at the same time, assigning jurisdiction to the Sofia (Bulgarian) court in one case and the Rome (Italian) court in the other. The system makes it possible to detect and point out this inconsistency, indicating that two different articles, with different answers in the matter of jurisdiction, should apply simultaneously.

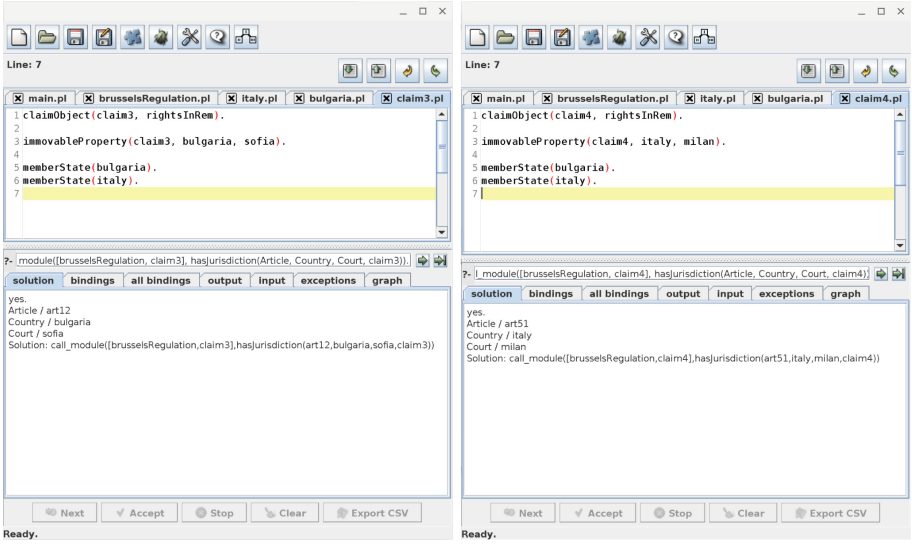

Fig. 2. Arg2p interface: result of claim3 (left) and claim4 (right).

Example 2 (Jurisdiction related to rights in rem). Let us now turn to the case discussed in Example 2. The facts of the case are stored in module *claim3.pl*. In particular, we have a fact establishing the object of the complaint (`claimObject`), in this case rights in rem. We then have a fact stating the place

of the immovable property in case (`immovableProperty`), which in the case at hand is Sofia. Finally, as in the example before, we have facts establishing which states are members of the EU—in our examples, Italy and Bulgaria.

```
claimObject(claim3, rightsInRem).

immovableProperty(claim3, bulgaria, sofia).

memberState(bulgaria).
memberState(italy).
```

To evaluate the case, we can select the jurisdiction simply by calling the following goal over the top module *brusselsRegulation.pl*:

```
call_module([brusselsRegulation, claim3],
        hasJurisdiction(Article, Country, Court, claim3)).
```

The result is shown in Fig. 2 (left) and states that the jurisdiction and the court must be in Bulgaria. Once again, the system shows that the connection between different modules is a necessary feature. In fact, since the immovable property is in Bulgaria, the Brussels Regulation defers Bulgarian legislation to determine the jurisdiction. Bulgarian law confirms that if the property is in Bulgaria, the case must be brought before a Bulgarian court. It should be noted that in this case the Italian law only states that if the property is not in Italy, then it is not for an Italian court to take the case, but nowhere does the law state which court *should* do so. Therefore, without such a connection between the different bodies of law, there would be no answer that the Italian court could offer in deciding where the case should be heard.

If we change the fact concerning the property, indicating that it is in Italy, the system will respond, correctly, that the case must be adjudicated by an Italian court, and in particular the Milan court (Fig. 2 (right)).

```
claimObject(claim4, rightsInRem).

immovableProperty(claim4, italy, milan).

memberState(bulgaria).
memberState(italy).
```

5 Conclusion

In this paper we have shown how the domain of private international law can be suitably modelled by modular argumentation with the support of Arg-tuProlog, which provides a way to reason about a formal model of interaction among multiple legal systems.

Indeed, we were aware that the PIL domain is particularly difficult to formalize, especially by comparison with domains that are traditional fields where knowledge-based systems in law are applied, such as tax, administrative, and entitlement law.

In the future, we aim to extend the modular-argumentation approach supported by Arg-tuProlog so as to cover additional parts of the domain of EU PIL law. In particular, we aim to cover international and national rules dealing with laws that apply to both contracts and non-contractual obligations and to extend the set of modules so as to cover a broader range of national legal systems.

In addition, we think that the same approach, that combines modular argumentation and defeasible reasoning, can be used to further model legal domains presenting characteristics and issues similar to PIL law (e.g., multiple levels of conflicting/overlapping rules), such as internet and aviation law.

References

1. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowl. Eng. Rev. **26**(4), 365–410 (2011). https://doi.org/10.1017/S0269888911000166
2. Besnard, P., et al.: Introduction to structured argumentation. Argument Comput. **5**(1), 1–4 (2014). https://doi.org/10.1080/19462166.2013.869764
3. Calegari, R., Contissa, G., Pisano, G., Sartor, G., Sartor, G.: Arg-tuProlog: a modular logic argumentation tool for PIL. In: Villata, S., Harašta, J., Křemen, P. (eds.) 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 December 2020. https://doi.org/10.3233/FAIA200880
4. Calegari, R., Omicini, A., Sartor, G.: Computable law as argumentation-based mas. In: Proceedings of the 21th Workshop "From Objects to Agents", WOA (2020)
5. Calegari, R., Pisano, G., Omicini, A., Sartor, G.: Arg2P: an argumentation framework for explainable intelligent systems. J. Log. Comput. **32**, 369–401 (2022). https://doi.org/10.1093/logcom/exab089
6. Calegari, R., Sartor, G.: Burden of persuasion in argumentation. In: Proceedings 36th International Conference on Logic Programming (Technical Communications), ICLP 2020. EPTCS, 18–24 September 2020. Camera-ready sent
7. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2), 321–358 (1995). https://doi.org/10.1016/0004-3702(94)00041-X
8. Dung, P.M., Sartor, G.: The modular logic of private international law. Artif. Intell. Law **19**(2–3), 233–261 (2011). https://doi.org/10.1007/s10506-011-9112-5
9. Modgil, S., Prakken, H.: The ASPIC$^+$ framework for structured argumentation: a tutorial. Argument Comput. **5**(1), 31–62 (2014)
10. Pisano, G., Calegari, R., Omicini, A., Sartor, G.: A mechanism for reasoning over defeasible preferences in Arg2P. In: Monica, S., Bergenti, F. (eds.) CILC 2021 - Italian Conference on Computational Logic. Proceedings of the 36th Italian Conference on Computational Logic. CEUR Workshop Proceedings, vol. 3002, pp. 16–30. CEUR-WS, Parma, 7–9 September 2021. http://ceur-ws.org/Vol-3002/paper10.pdf

11. Prakken, H.: An abstract framework for argumentation with structured arguments. Argument Comput. **1**(2), 93–124 (2010). https://doi.org/10.1080/19462160903564592
12. Riveret, R., Rotolo, A., Sartor, G.: A deontic argumentation framework towards doctrine reification. J. Appl. Log.–IfCoLog J. Log. Their Appl. **6**(5), 903–940 (2019). https://collegepublications.co.uk/ifcolog/?00034
13. tuProlog: Arg-tuprolog repository. https://github.com/tuProlog/arg2p-kt
14. tuProlog: Arg-tuprolog website. https://pika-lab.gitlab.io/argumentation/arg2p-kt/