Legal Knowledge Representation in the domain of Private International Law*

Giuseppe Contissa[1] and Galileo Sartor[2]

[1] Alma Mater Research Institute for Human - Centered Artificial Intelligence (Alma Human AI), University of Bologna, Italy giuseppe.contissa@unibo.it
[2] University of Turin, Italy galileo.sartor@unito.it

Abstract. This paper presents the development of a Prolog rule-based system in the domain of Private International Law. After having identified the legal and technical requirements for the representation, methodological choices and issues encountered during the development are discussed. Then, an example of the functioning of the system is presented. Finally, results are discussed.

Keywords: Legal Knowledge representation · Private International Law · Prolog.

1 Introduction

This paper presents the development of a Prolog rule-based system in the domain of Private International Law. The research has been carried out in the context of the European Project Interlex, whose aim is to develop an advisory and training system for internet-related private international law, and to make it available as an online platform. The platform will be composed of three modules: The Decision Support Module (DSM), the Find Law Module (FLM) and the Training Module (TM). In this context, the Prolog representations will be the core component of the Decision Support Module, within which it will provide basic legal reasoning capabilities.

The paper is organised as follows: in section 2 we present some legal and technical requirements that were taken into account when developing the representation, and the legal domain that was represented. In section 3 we present the structure of the developed representation, and the approach used to model legal knowledge and reasoning. Then, in section 4, we demonstrate the use of the rulebase trough an example of private international law. In section 5 we present our conclusions and some thoughts on future extensions of the system.

* This work has been partially 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".

2 Methodology

2.1 Requirements for legal knowledge representation

In this section we will cover the necessary requirements for modelling legal norms. These are strictly tied to how we write and express legal norms in natural language. Among the various requirements identified by experts for legal knowledge representation [GGR09], we consider some of them particularly relevant for our aims:

- isomorphism: one to one correspondence between norms in the formal model and natural language. In Karpf [Ka89] we find five conditions that have been listed in Bench-Capon and Coenen [BC92] as the following: (i) Each legal source is presented separately; (ii) The representation preserves the structure of each legal source; (iii) The representation preserves the traditional mutual relation, references and connections between the legal sources; (iv) The representation of the legal sources and their mutual relations (...) is separate from all other parts of the model, notably representation of queries and facts management;
- reification: rules representing legal norms need to be treated as object with properties by other rules, in order to deal with the following aspects of legal systems and norms.
  - Jurisdiction: that is the limits within the rules are authoritative and binding.
  - Authority: that is the property specifying who produced the rule, and thus it indicates its ranking within the sources of law (e.g. constitutional provision, statutory law, regulation, administrative, etc.).
  - Temporal properties: rules are usually qualified by temporal properties, in particular the internal and external time of a norm [PGC10].
- defeasibility: when the antecedent of the rule is satisfied by the facts of the case, the conclusion of the rule presumably holds, but this presumption can be defeated. The defeasibility of the legal rules comes down to the following issues:
  - Conflicts: rules may lead to incompatible legal effects, that can be resolved through rule priorities, such as: lex specialis (which gives priority to the most specific rule), lex superior (which gives priority to the rule from the higher authority), lex posterior (which gives priority to the rule enacted later).
  - Exclusionary rules: rules that provide a way to explicitly undercut other rules, making them inapplicable.
- the ability to deal with vague concepts, either by replacing them with a strict variant (for example the concept of good faith can be presumed to be true [Se86]) or by presenting the different possible outcomes.

Moreover, a set of additional requirements concerns the formal language to be adopted, both from the technical and the legal perspectives:

- Logic aspects and formal semantics
  - Basic syntactic elements, operators, rules of combinations. It covers (1) a description of the basic syntactic expressions of the language; (2) operators, including temporal and deontic operators; (3) rules of combination (which define all and only the well-formed expressions of the language); rule recursiveness.
  - Constraints on the Heads or Bodies of Rules. Rules may allow complex propositions in the head or body, formed by atomic propositions.
  - Expression of Negation, Disjunction, Conjunction. In particular, how the language deals with negation by failure, and logic negation.
  - How values are assigned to variables. Variables may have an undefined value, preventing us from determining the truth values of complex expressions. It is then left to the user (or a system internal operation) to assign a value. As the values of individual variables are determined, the values of the propositions containing the variables are correlatively determined.
- Defeasible Rules and Exceptions: languages may treat defeasibility on norms in different ways: establishing priorities between rules, or with rules explicitly defeating other rules.
- Tractability and computational complexity: how to measure the extent to which the language is capable of capturing the structure of the norm, in a way that it easily readable and modifiable.
- Justifications: the feasibility and the extent to which the language can express the reasons supporting the conclusion in human readable and understandable form.
- Extendibility: the possibility of the language to be extended with web interfaces, API, external modules, etc.
- Interoperability: the feasibility and the extent to which the language can interoperate with existing standards for expressing legal knowledge and reasoning (LegalRuleML).
- Portability on different platforms.
- Development support: Availability of IDEs, supporting tools (tracers, debuggers) and documentation.
- Licensing: availability under open source licence.

In an initial analysis and testing phase multiple languages were tested, in order to identify which one managed to support a complete representation of the previously mentioned characteristics, without a burden too heavy on the development and expansion of the rulebase. The languages tested were Prolog (in the SWI-Prolog variant[3]), Oracle Policy Automation[4], and Turnip[5]. Each has its own strengths, and is better suited for a specific workflow. Oracle Policy Automation is a simple to use graphic tool, that is very visual in its representation of norms, albeit a bit limited in the possibility of expansion outside of the

[3] `https://www.swi-prolog.org`
[4] `https://www.oracle.com/applications/oracle-policy-automation/index.html`
[5] `https://turnipbox.netlify.app`

predefined tags. Turnip is built from the ground up for representing norms, so for instance it includes ways to represent defeasible norms and exceptions natively. It can be however more complex to read and to maintain a codebase outside of the primary intended goal, that is to enable the use of a shared database containing the facts of the representation. Prolog is more of a blank sheet, being a generic logic programming language, not built specifically for the purpose of this project. This can be very useful, and with a more involved and complex planning phase, can simplify the model used down the line, since it can be tailored to the needs of the knowledge engineer that will maintain it and update it in the future.

2.2 Technical requirements

In this section we will identify how the legal requirements can be translated in technical specifications for the system and language. In the context of the Interlex project the main specifications that must be assessed are:

- Tractability and reasonable computational complexity: A problem is called intractable if the time required to solve instances grows exponentially with the size of the instances. It is important because exponential growth means that even moderately large instances cannot be solved in any reasonable time. Therefore, one should strive to divide the overall problem of generating intelligent behaviour into tractable sub-problems rather than intractable ones
- Semantic values of the expressions: The basic values that can be assigned to propositions are "true" and "false", but in some domains it might not be enough, and there might be a need to address uncertainty, and the possibility of an "unknown" value.
- Control strategies: To adequately implement the knowledge base of a rule system, we must consider how the rules are used. The answer to this question can seem easy, but it can change the bulk of the representation.
- Explanation and justification: Explanation in rule-based systems is usually associated with a way of tracing the execution of the rules that are fired during the course of a problem-solving session. This is about the closest to a real explanation that today's systems can provide, given that their knowledge is usually represented almost exclusively as bare rules and does not include basic principles necessary for a human-type explanation. Explanation is an extremely important function because understanding depends on explanation, and makes it easier to implement proposed solutions easier. constructing explanations can become a very complex task, especially when undertaken by machines.

2.3 The legal domain

The rulebase representation covers the three main EU legislative instruments on Private International Law (PIL).: the Regulation (EU) No 1215/2012 on jurisdiction and the recognition and enforcement of judgments in civil and commercial

matters (recast) (Brussels Regulation); the Regulation (EC) No 593/2008 on the law applicable to contractual obligations (Rome I); and the Regulation (EC) No 864/2007 on the law applicable to non-contractual obligations (Rome II).

The reference documents adopted as sources for the representation were the digital copies in pdf format and in English language available on the eur-lex website. The development of the representation started with a legal analysis of the EU regulations, aimed at identifying the core part of laws to be represented in relation to the goals of the InterLex project.

As a general rule, the articles relevant for the assessment of jurisdiction and applicable law have been represented. The missing articles contain either clarifications or examples (e.g. non-closed definitions). The representation of these rules will be evaluated for future versions of the rulebase.

Then, each norm from the core parts of the legal source texts was represented in a correspondent rule (or set of rules) by specialised knowledge engineers supported by legal experts. The legal analysis often involved the interpretation of complex legal rules, and the selection of one interpretation among several possible. Such analysis was carried out with the support of legal commentaries, namely [07] and [Ca15].

No deviations were made, as long it was possible, from the original structure of the text, even when it was redundant or convoluted (as regard to the logic representation), in order to preserve isomorphism (see section 2 above).

3 The Prolog representation

3.1 High-level structure of the rulebase

The language adopted for the representation is SWI-Prolog, according to the selection carried out at the beginning of the project and documented in section 2. However, the similarity of SWI-Prolog implementation to the ISO-Prolog standard also means that the Prolog code is not entirely specific to the chosen Prolog implementation, but it can be easily re-implemente with other versions of the language.

The Interlex RuleBase base is logically organized according to the specific chapters and sections of the source legal documents that have been selected for the representation. This is mirrored in the code base by having the different sections split in separate files. Each legal source (Brussels Regulation, Rome I Regulation, Rome II Regulation) has its own entry point/query, respectively:

```
hasJurisdiction(Country, Court, ClaimId, Law)
applicableLaw(Article, Country, ContractId, Law)
applicableLawNonContract(Article, Country, ObligationId, Law)
```

The top level goal is immediately followed by the main exceptions and conditions, that must be verified before importing the rules contained in other chapters and sections. As an example, this is a fragment from the Brussels Regulation representation:

```
hasJurisdiction(Country, Court, ClaimId, brusselsRegulation):-
brusselsRegulationApplies(ClaimId, brusselsRegulation),
\+exception(hasGeneralJurisdiction, _, ClaimId),
hasGeneralJurisdiction(Country, Court, ClaimId, brusselsRegulation).
```

In this example, the top-level goal (the assessment of the jurisdiction according to the Brussels Regulation) can be verified if the premises (contained in Chapter 1 of the Regulation) apply, there are no exceptions to the general jurisdiction rules , and at the same time the general jurisdiction rules apply. The same approach has been adopted in the representations of the Rome I and Rome II Regulations.

3.2 The approach to the modelling

The conversion process The process of conversion of the initial legal text into rules has necessarily undergone several different stages, as required by commonly adopted Prolog-based rule-base creation processes. This work was carried out following a sequence of iterative stages: the first stage has consisted in the development of what is usually called the Pseudo-Code, that is the rewriting of the initial legal text in an intermediated controlled natural language (English), that makes explicit logical structures and connectors, while keeping as much as possible of the original semantics. This is the stage that has necessarily to be supervised by legal experts. During this stage specific guidelines has been followed, taking into account further processing of representation. More in detail:

1. the knowledge engineer has tried as much as possible to keep all the original sentences in such a way that a rule (or more rules) matches a sentence or expanded sentences;
2. the connectives (or, and, etc.) have been developed by repeating the relevant part of sentence or creating a new sentence;
3. the relatives (which, who, etc.) have been developed and explicated;
4. pronouns have been replaced by their referred nouns or lexical expressions;
5. irrelevant terms and repetitions have been suppressed;
6. when necessary, sentences have been to get simpler syntax sentences;
7. whenever possible, use has been made of equivalent terms to minimize the number of derived predicate names;
8. sequences of words have been replaced by single terms whenever compound words were found.

In the second stage of the work, we have adopted a one-step processing which, starting from the "pseudo-code", reached up to the formalization in Prolog rules. This stage has been broken into three successive steps: Step 1 – On the basis of an initial analysis of the list of nouns, verbs and adjectives contained in the source law texts, relevant lexical terms have been identified, and a basic taxonomy has been developed of the concepts of PIL, their properties and relations. Step 2 – Then, it has been evaluated whether to include them as Prolog predicate names or arguments. Since a lexical term can be formalized as a different object type in the logical rules: predicate with different arities, function, constant, etc.,

the taxonomy has been used, and when necessary updated, to maintain consistency during the translation of the whole corpus of law texts. Step 3 - The final formalization has been carried out from pseudo-code to Prolog rules. We have adopted the methodology described in section 2. We have also decided to introduce structural elements in the rulebase, in the form of predicates, to refer to corresponding structural elements of the source legislative text (chapters and sections). Consider for example the following fragment of rules, extracted from the representation of Brussels regulation, section 7:

```
hasJurisdiction7_1to7(Country, Court, ClaimId, brusselsRegulation):-
    hasJurisdiction7_1(Country, Court, ClaimId, brusselsRegulation);
    hasJurisdiction7_2(Country, Court, ClaimId, brusselsRegulation);
    hasJurisdiction7_3(Country, Court, ClaimId, brusselsRegulation);
    hasJurisdiction7_4(Country, Court, ClaimId, brusselsRegulation);
    hasJurisdiction7_5(Country, Court, ClaimId, brusselsRegulation);
    hasJurisdiction7_6(Country, Court, ClaimId, brusselsRegulation);
    hasJurisdiction7_7(Country, Court, ClaimId, brusselsRegulation).

hasJurisdiction7_1(Country, Court, ClaimId, brusselsRegulation):-
    claimObject(ClaimId, contract, ContractId),
    placeOfPerformance(Country, Court).

hasJurisdiction7_2(Country, Court, ClaimId, brusselsRegulation):-
    claimObject(ClaimId, tort),
    eventOccurredOrMay(Country, Court).
```

In this example, the introduction of predicates in the form hasJurisdiction7_1, 7_2, 7_3, etc), has a twofold purpose: 1) to maintain isomorphism of the representation, keeping a close connection between the structure of the source texts and the structure of the representation, and 2) to keep trace of the articles in the execution process, so as to simplify debugging and validation of the rulebase, and even more important, to improve the quality of explanations provided by the meta-interpreter (see the technical requirements in section 2). The same approach has been adopted for the Rome I and Rome II Regulations, with the inclusion of an additional argument, Article, that is assigned with the article that verifies the Goal. For consistency the article reference in the predicate name was kept in the Rome Regulations, and the Brussels representation will be updated. In the code comments are used mostly as references for internal development processes. Prolog can also build a documentation html structure from a subset of the documentation, so some testing has been done to keep the original text embedded in the source code, to be able to view the relevant code and legal text swiftly. The main testing for the codebase is based on a small number of cases, and is automated via a python wrapper.

Defeasibility and exceptions It is commonly agreed that legal reasoning is a defeasible reasoning. In EU PIL corpus, defeasibility occurs by means of expressions such as: “unless proved otherwise”; “unless otherwise agreed”; “unless ...”; “except in those cases”; “with the exception of”; “subject to...”; “notwithstanding”; etc. In Prolog, defeasibility is usually managed by using negation by failure. Hence, each of the above items has to be accounted by using negation by failure.

Non-ISO: Meta interpreter The main non-ISO structure in the rulebase code is the meta-interpreter. As previously explained, a Prolog meta-interpreter

is a Prolog program, that takes a Prolog goal and another Prolog program, then proceeds to attempt to prove the goal against the second Prolog program, according to the rules given by the first one. At a simple level the meta-interpreter could be simply prove(Goal) :- call(Goal), but this would not give any more information, it would simply call a goal to prove it. In the following we present the implemented version of the meta-interpreter developed in the context of the project. We built this simple (but expandable) meta-interpreter to generate a proof tree tracing the execution of the program, by printing a log of the evaluated predicates, as a first attempt at providing a user-friendly explanation functionality. Another possibility that derives from having a meta-interpreter is the ability to interact with the user, via askable goals, goals that can be asked directly to the user if the system finds missing facts. This can be used to solve the issue of missing undefined values expressed previously. In the execution of a goal, the asserted predicates are logged as facts, to simplify the readability of the produced document.

```
% Metainterpreter
solve(true, [fact]) :- !.
solve((A,B), Result) :- !, solve(A, ARes), solve(B, BRes),
                        append(ARes, BRes, Result).
solve((A;B), Result) :- solve(A, Result); solve(B, Result).
solve(member(A,B), [member(A, B)]) :- !, call(member(A,B)).
solve(\+(A), [not(A)]) :- !, call(\+(A)).
solve((A)\=(B), [doNotUnify(A, B)]) :- !, call((A)\=(B)).
solve(A, [A|[Res]]) :- clause(A,B), solve(B, Res).
```

The output of the meta-interpreter can be integrated easily in other systems/programs. In our case we developed a small program in Python to format the output in a readable fashion, and to enable the remote use of the Prolog program via a web api. Another possible use of this simplified integration is the ability to automatically build a document explaining in user-friendly terms the reasoning of the program.

4 Use of the rulebase

In this section we present an example showing the use of the SWI-Prolog engine in combination with the rulebase and a set of facts representing a hypothetical scenario involving the application of EU rules of Private International Law.

Use case scenario: Provision of services

Let us consider the following hypothetical scenario: Silva Trade, a (natural) person located in Luxemburg, signs a contract with WoodFloor, a company located in Turin, Italy: Woodfloor agrees to provide its services to Silva, in the cities of Vienna and Paris, and Silvia agrees to pay the fee for the services. However, services are not provided as expected by Silva, so that she refuses to pay the fee. Woodfloor decides to sue Silvia. The question to be assessed is: which court has the jurisdiction for the claim? In Prolog, the above facts and the query may be represented as follows:

```
assert(claimMatter(claimId7_1, civilCommercial)).
assert(claimObject(claimId7_1, contract, contractId7_1)).
assert(contractTypeConsideration(contractId7_1, consumer)).
```

```
assert(contractType(contractId7_1, provisionOfServices)).
assert(personDomicile(silvaTrade, luxemburg, _)).
assert(personDomicile(woodFloor, italy, turin)).
assert(personNature(silvaTrade, natural)).
assert(personNature(woodFloor, legal)).
assert(personRole(silvaTrade, claimId7_1, defendant)).
assert(personRole(woodFloor, claimId7_1, claimant)).
assert(personType(silvaTrade, consumer)).
assert(placeOfProvision(austria, vienna)).
assert(placeOfProvision(france, paris)).

hasJurisdiction(Country, Court, claimId7_1, brusselsRegulation).
```

The Prolog system provides three alternative solutions for the query:

```
luxemburg, _ ; france, paris ; austria, vienna
```

In order to verify the top-level goal of this chain (hasJurisdiction(luxemburg, _1596, claimId7_1, brusselsRegulation), Prolog has to verify its main conditions. The meta-interpreter collects such conditions at the level just under the top-goal, as part of a nested list:

```
[hasJurisdiction(luxemburg, Court, claimId7_1, brusselsRegulation), [
    brusselsRegulationApplies(claimId7_1, brusselsRegulation), [not(
    exception(brusselsRegulationApplies(claimId7_1, brusselsRegulation),
    9548)), claimMatter(claimId7_1, civilCommercial), [fact]], not(
    exception(hasGeneralJurisdiction, 9484, claimId7_1)),
    hasGeneralJurisdiction(luxemburg, Court, claimId7_1, brusselsRegulation
    ), [personRole(silvaTrade, claimId7_1, defendant), [fact],
    personDomicile(silvaTrade, luxemburg, Court), [fact], memberState(
    luxemburg), [fact]]]]
```

Such initial trace provided by the meta-interpreter is then formatted by a simple python wrapper as follows:

```
  TOP GOAL: luxemburg, 1
-> hasJurisdiction(luxemburg, _1596, claimId7_1, brusselsRegulation)
1)   -> brusselsRegulationApplies(claimId7_1, brusselsRegulation)
         -> not(exception(brusselsRegulationApplies(claimId7_1,
                                  brusselsRegulation), _1710))
         -> claimMatter(claimId7_1, civilCommercial) -> FACT
2)   -> not(exception(hasGeneralJurisdiction, _1646, claimId7_1))
3)   -> hasGeneralJurisdiction(luxemburg, _1596, claimId7_1,
                               brusselsRegulation)
         -> personRole(silvaTrade, claimId7_1, defendant) -> FACT
         -> personDomicile(silvaTrade, luxemburg, _1596)  -> FACT
         -> memberState(luxemburg) -> FACT
```

In this explanation, the Prolog correctly verified the main goal, with the value country= Luxemburg, and court= unknown (_). The system then displays the predicates it verified to reach that goal (identified as 1), 2), and 3) in the code lines above), indented to represent their logical chaining. The same happens to every intermediate goal that may need to be verified (in this case, the sub-goals are those identified as 1), 2), and 3), and their immediate sub-sub-goals). The bottom-level statements that are verified by the Prolog engine correspond to the facts asserted at the beginning of the example.

5 Conclusions

The complexity of source norms has sometimes limited the isomorphism of representation, in particular for the management of norms constituting implicit ex-

ceptions of general norms. However, the issue has been solved with the solution proposed in section 3 and applied as in section 4 of the present work. Besides, we were aware that the domain of PIL would be particularly difficult to formalize, especially when compared with those domains that are traditional fields of application of knowledge-based systems in law, such as tax legislation, administrative regulations, provision of benefits, etc. On the one hand, tax legislation is commonly considered "complex" because of the high number of exceptions and requirements to be verified to reach a conclusion, but tax rules are nevertheless quite easy to be represented because they do not need any particular work of interpretation by a legal expert. On the other hand, the PIL domain contains a very high number of concepts which lack a definition set by the lawmakers, and which may therefore be considered as "open texture" concepts (as defined in legal philosophy by Hart [Ha61] and discussed in relation to legal information systems by [BV97]), and their meaning should be identified searching in a multiple domain context and taking into account case law and doctrinal interpretations. Despite the high difficulty and complexity of representation of the PIL law, the results present in this report show how potentially any field of the law may be represented in form of rules, using the adopted methodology – with the opportune adaptations – and transposed into a platform such as InterLex.

References

1. Bench-Capon, T. J. M.; Visser, P. R. S.: Open texture and ontologies in legal information systems. Database and Expert Systems Applications. 8th International Conference, DEXA '97. Proceedings/, 1997.
2. Bench-Capon, T.; Coenen, F.: Isomorphism and legal knowledge based systems. Artificial Intelligence and Law **1**/1, 65–86, 1992.
3. Brussels I Regulation (European Commentaries on Private International Law). sellier european law publishers, 2007, ISBN: 978-3-935808-32-3.
4. Calliess, G.-p.: Rome Regulations: Commentary on the European Rules of the Conflict of Laws. Wolters Kluwer Law & Business, 2015, ISBN: 978-90-411-4754-7.
5. Gordon, T. F.; Governatori, G.; Rotolo, A.: Rules and norms: Requirements for rule interchange languages in the legal domain. In: Rule interchange and applications. Springer, 282–296, 2009.
6. Hart, H. L. A.: The concept of law. 1961.
7. Karpf, J.: Quality assurance of Legal Expert Systems. Jurimatics **8**/, 1989.
8. Palmirani, M.; Governatori, G.; Contissa, G.: Temporal Dimensions in Rules Modelling. In: Legal Knowledge and Information Systems JURIX 2010: The Twenty-Third Annual Conference. IOS Press, 2010.
9. Sergot, M. J.; Sadri, F.; Kowalski, R. A.; Kriwaczek, F.; Hammond, P.; Cory, H. T.: The British Nationality Act as a Logic Program. Communications of the ACM **29**/5, 370–386, 1986.