Do Lawyers use Automated Reasoning?

Tomer Libal

University of Luxembourg, Luxembourg

Abstract

Laws must be understood and their application must be explained and justified. As such, they seem to be the perfect use case for (non-classical) automated reasoning. Indeed, there is much academic research in legal knowledge representation and reasoning. Nevertheless, there are no commercial reasoning-based applications, apart from legal expert systems. In this paper, we review some of the problems towards this goal and describe a possible solution.

Keywords

Automated Reasoning, Formal Representation, Knowledge Validation, Legal Informatics

1. Introduction

Information systems are playing an important role in helping people in a wide range of tasks, ranging from searching to decision-making.

One area in which such tools can contribute is the legal domain: New court cases and legislations are accumulated every day. In addition, international organizations like the European Union are constantly aiming at combining and integrating separate legal systems [1].

Approaches for searching over legal texts have long seen commercial success [2]. At the same time, some tools for reasoning over sets of norms have been developed, such as for business (e.g. [3]) and law (e.g. [4] and [5]). Among the applications of such tools are legal drafting, consistency checks and for deducing implications [6].

The most important application for these tools, though, is for capturing expertise. Expert systems [7] have been successfully utilized in various domains, among them in finance [8] and health [9].

Nevertheless and despite their proven usefulness, expert systems are not widely used in the legal domain. The main reason for that is the difficulty to capture the expert knowledge within a computer program [10]. The two mains reasons for this difficulty are the requirement to have two domains experts: programmer and knowledge, as well as the need to transform the knowledge into a form which can be processed by a machine. An example for the former is the need to have a medical doctor and a programmer for

ARQNL 2022: Automated Reasoning in Quantified Non-Classical Logics, 11 August 2022, Haifa, Israel
EMAIL: tomer.libal@uni.lu (T. Libal)
URL: https://tomer.libal.info (T. Libal)
ORCID: 0000-0003-3261-0180 (T. Libal)

the creation of a medical expert system, while for the second, the need to transform the knowledge into rules, in order to be able to present it to the user.

The second challenge is especially important as it affects the trust that can be placed in the expert system. The more processing which is required, the less trust that can be placed in the knowledge. Even systems which are created with a relatively flexible tool, such a programming language, suffer from knowledge transformation. In their seminal paper, Sergot et al. [11] have shown how a legal expert system can be created by using the Prolog logical programming language. Others (e.g. [12]) have pointed to the various problems and mainly to the need to further interpret knowledge in a format supported by the programming language.

Both these challenges have been been the focus of research. The need of several domain experts is dealt with mainly via no-code platforms, such as Neota Logic [13] and Visirule [14]. Nevertheless, they do not solve the second challenge and are therefore hard to validate and maintain.

The second challenge is addressed by providing a richer language in which to capture knowledge. Such systems are based on a reasoning engine, which tries to emulate actual legal reasoning, and normally depend on non-classical logics such as Defeasible Deontic [15] or Higher-order logics [16].

These systems have the potential to overcome the main disadvantages of expert systems. By utilizing a more sophisticated reasoning engine, they allow for a more compact and faithful representation of the original legislation, while preserving the main advantages of expert systems.

Nevertheless, such systems have enjoyed little commercial success in the legal domain. **The main goal of this paper is to investigate the reasons for that and to suggest remedies**.

The most obvious shortcoming of such systems is their inability to address the first challenge. In fact, often, the richer the language is, the higher the need is for several domain experts.

This usability issue was identified to be one of the main requirement of expert systems building tools (e.g. [17]). Nevertheless, most such tools fail in achieving this goal [18].

Various attempts have been done in order to make such tools more accessible to legal practitioners via a more user-friendly interface. Logical English for example [19], provides an English based controlled natural language interface, which communicates with the underlined logic program. The advantage of these approaches is the ability to compile and test the correctness and quality of the knowledge.

Another approach, demonstrated in [20], is by the use of general formats and languages. Being of general formats, one can more easily utilize a language which is most suited for capturing the semantics of legal texts. One disadvantage of this approach is that these formats do no directly give raise to automated reasoning.

A main additional disadvantage of all the approaches above is the lack of tools and methodologies for asserting the correctness of the logical representations of the legal texts. A discussion about the need can be found in [21][1].

[1] See discussion in Sec. 3.2 in [21]

Among existing validation results, one can find a methodology for building legal ontologies [22] and more concretely for building legal knowledge bases, one for validating formal representations of legal texts [23].

To summarize, the above approaches underline a key issue in legal reasoning and knowledge representation, which is the tension between the expressiveness of the knowledge and the efficiency of reasoning over it [24]. Expressivity refers to the ability of the language to directly capture the nuances of the target language, which in our case is the legal language. Being efficient to reason over refers to the ability to use existing and efficient tools to reason over formulae denoted in this language.

Those approaches which use a programming language as the format for knowledge representation enjoy efficient reasoning but require a non-trivial translation in order to capture the nuances of the law. Moreover, while they are often capable of producing a proof, these proofs are not easily translated into legal arguments. On the other hand, approaches targeting the legal text directly, do not offer a direct reasoning engine.

In the next section, we focus on this tension and offer a possible solution - the use of two different languages, one expressive and one efficient - and an automatic bidirectional translation mechanism between them. This section builds on the works in [25] and [26]. The following section describes an implementation of this approach. We then conclude we a summary of the results and some future work.

2. Legal Linguistic Templates

This section is based on the meta-level annotations introduced in [25], which are further developed into LLTs in the joint work in [26].

As discussed in the previous section, legal formalisms suffer from the tension between expressiveness and efficient reasoning. This tension is captured by the following three requirements [27]:

1. be a faithful representation of what is expressed by the legislation
2. be computationally adequate, i.e. should permit us to make all relevant derivations by machine
3. be easy to validate and maintain.

General and extensible formats such as DAPRECO [20] can clearly achieve point (1) above. But obtaining point (2) seems challenging. For programming languages, the opposite is true, as they are computationally adequate by definition. On the other hand, they require a translation between the original text and the programming language and might, therefore, not be a faithful representation.

Both approaches would struggle with point (3). On the one hand, we expect the formalization to be certified by a legal expert. On the other, both programming languages and other formal formats require a programmer or a logician for the encoding. This gives raise to methodologies for validation such as [23].

Maintaining the knowledge base requires the ability to track changes in the law and apply them to the formalization. Any encoding of the law which cannot be easily traced

back to the original text would not be easily maintainable. The maintaining property can be directly linked to the need of an isomorphic translation, which maps a one-to-one relation between the original text and its formal version. Such isomorphism allows mapping back from the formal version to the original version and therefore to identify the elements in the formalism which would require a change. If the encoding requires a translation which is not isomorphic to the original text, tracing the result back would not be trivial.

It seems then than none of the current systems can support more than one of the requirements.

We propose a new approach for defining new systems, which follows the above reasoning. We identify point (3) above as the one currently not adequately supported by existing systems. We then propose a solution which meets the requirements of this point and develop the idea further in order to support the other two points.

An important point which is not directly covered by the above three points, but which is often mentioned in discussions with lawyers, is accountability. We consider this point as closely related to validation, as it qualifies what is a good validation process.

We will next introduce the basic idea of such an expressive language and claim that it can both represent faithfully the original text, as well as be easily maintainable.

We then go on to argue that it satisfies the different required properties.

If we consider the formalization processes as a function from legal texts to logical formulae, the maintainability requirement states that this function must be isomorphic. Given a formula, we should be able to generate the original text.

It seems that the only possible solution to that is to use a logic which is as expressive as the original text. This also implies point (1). If the logic is as expressive as the original, then regenerating the original text amounts to just pretty printing formulae. Using a logic of lower expressivity would cause any translation into original text to result in some level of information loss.

In order to define such a logic, we first need to understand what we try to formalize.

Definition 1 (Legal interpretation)**.** *A legal interpretation of a legal text is defined as the interpretation of a legal expert with regard to a specific context problem. It should be noted that interpretations are usually subjective and context dependant.*

In order to formalize legal interpretations in a formal language, we propose using the Legal Linguistic Templates (LLTs) language. This language is a general logical language and contains an interpreted logical connective for each foundational legal construct. Examples of such constructs are prohibitions and exceptions. The basic elements of the language are first-order atoms.

Using the above language, we can define formal interpretations of legal ones.

Definition 2 (Formal interpretations)**.** *Given a legal interpretation L containing n statements, a formal interpretation for L is a set of n statements over an LLT language.*

Before we move forward to the computational adequacy of the LLTs language, we would like to mention that within the terminology of programming languages [28], a

formal interpretation can be reduced into an abstract syntax tree over a first-order term language.

By choosing an LLTs language which is expressive enough to describe a “good” formal interpretation of a legal text, we seem to go further away from computational adequacy. We now show that this is not necessarily the case.

We achieve that by following the "shallow semantical embeddings" defined by Benzmüller [24]. The basic idea of this approach is to provide “a lean and elegant equational theory which interprets the syntactical constituents” of logic L (in our case a specific LLTs language) as terms of another logic M.

If the logic M that we choose is computationally adequate, then we effectively obtain the computational adequacy of L. This is achieved in practice by applying the shallow semantical embedding, and then applying software for the computationally adequate result.

The logic chosen by Benzmüller is Higher-order Logic [29] and the various software which can be used to reason over the embeddings include Isabelle/HOL [16] and Leo-III [30].

The LLTs languages we have used so far seems to be too general for the use of shallow semantical embeddings. We will next generalize the notion of a "shallow semantical embedding".

Definition 3 (Embedding functions)**.** *An embedding function ϕ of a logic L into a logic M, is a total function from L to M.*

Definition 4 (Adequate meta-logics)**.** *An adequate meta-logic is a computationally adequate logic, into which an embedding function from another logic exists.*

For example, first-order logic with negation interpreted as negation-as-failure (NAF) [31] is computationally adequate as embeddings in this language can be executed in Prolog [32].

Assuming that our LLTs language contains a connective for prohibition and another for exceptions, we can describe the following embedding function into a first-order logic with NAF.

- $\phi(\emptyset) \Longrightarrow \emptyset$
- $\phi(S \cup \{\texttt{GeneralProhibition(B)}\}) \Longrightarrow \texttt{prohibited}(\phi(\{B\})) \cup \phi(S)$
- $\phi(S \cup \{\texttt{Exception(A,B)}, \texttt{LabeledStatement(A,C)}\}) \Longrightarrow \{\texttt{not } \phi(\{B\}) \Rightarrow \phi(\{C\})\} \cup \phi(S)$

Note that the resulted set denotes a set of first-order formulae. A standard normalization can be used in order to obtain a logic program where arguments of atoms are universally quantified on the level of the whole program. The function symbol `prohibited` is uninterpreted.

Having the embedding, we can now use Prolog in order to compute various properties. To check for violations of the prohibition, for example, one would need to add Prolog facts and then check if `prohibited(_)` can be derived.

The last requirement of a legal formalization language is to have a good validation process, which can ensure that legal experts be held accountable.

The approach taken in order to meet this requirement follows a very basic idea in programming languages, that of a pretty printer [33]. Pretty printing takes a formal structure, such as an LLT formula, and presents it in a user friendly form.

While the concept is basic, it turns out to be very powerful in the legal domain. By printing an LLT formula as a legal statement in English, we effectively translate it back into the original legislation language. This feature allows a legal expert to compare the two versions of the legislation - the original and the pretty printed one - and to make a decision regarding its validity. This decision can be made even when the legal expert does not have any knowledge in logic and has not participated in the generation of the LLT formulae.

3. The LegAi editor - an LLT implementation

In this section we describe an implementation of the LLTs language. The goal of the implemented LegAi annotation editor is to support annotating legal texts with LLTs. At the same time, the editor supports the validation mechanism described earlier in the paper by providing legal experts with a comparison of the original and generated text.

The editor can be found online[2] and requires a registration. After registration, the editor displays a message that an email to activate the account must be sent to the code maintainer, who associates the new user to a specific account with specific rights.

Once logged in, the user has the ability to paste legal texts into an annotation editor. The user starts with selecting a sentence they wish to annotate and then a (top-level) LLT wizard opens. The wizard guides the user in the annotation process, shows the possible/required LLTs that can/must be nested.

The LLT language contains one "leaf" LLT, called Atom, which can be instantiated with all possible vocabulary. The wizard ensures that annotations are always syntactically correct.

When selecting a specific vocabulary, the wizard allows selection of all parameters, according to their type. The user has control over how to generate or choose variables for the vocabulary and its parameters.

Once a top-level LLT was created, the user can click on the annotation and see a tree structure corresponding to the formal version of the text.

A list of all vocabulary and their associated legal texts appear on a separate tab.

An example of annotating two simplified articles of the GDPR is shown on the top of Fig. 1.

In order to check if an annotation captures correctly the meaning of a legal sentence, the user can view the formal tree structure, as defined in the previous section and shown on the middle of the figure.

Nevertheless, such a formal structure is not easily readable by non-logicians. In order to rectify that, a feature called reverse translation, which implemented the pretty printer

[2] https://legai.uni.lu

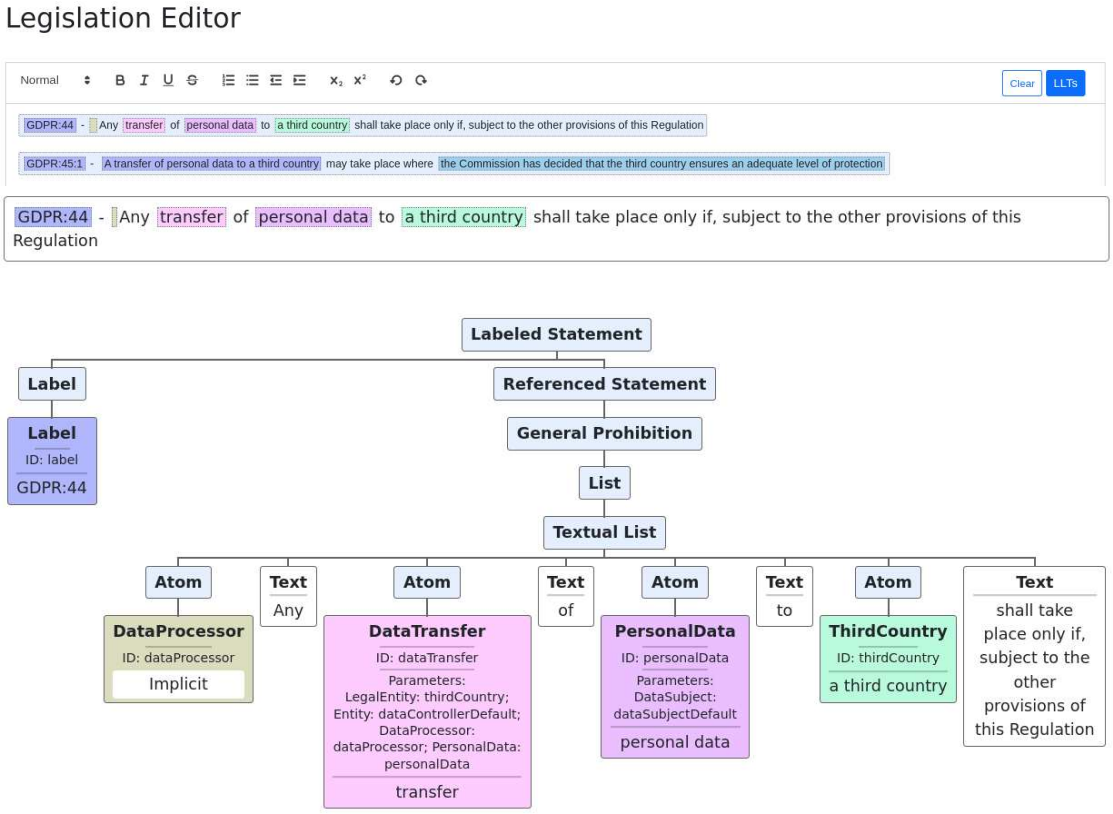

Figure 1: The annotation of article 44 of the GDPR.

approach, was added to the editor.

The reverse translation, which is accessible on a separate tab, generates a legal text from the formal tree structure. It then places the reversely generated text next to the original text. The user can then compare the two versions and decide if the formal version correctly captures the meaning of the original one, as shown on the bottom of the figure. Since our vocabulary is denoted via first-order terms, each contains both a name and a set of parameters. In the reverse translation, we display the name only, while hovering over the text shows all the used parameters.

4. Conclusion

In this paper we have considered the creation of formal interpretations of legal texts. This problem is one of the main ones preventing automated reasoning from being more utilized and commercialized.

We have discussed the main current approaches and identified several requirements which need to be addressed. We then defined a language, called LLTs, which we argue satisfies all the requirements.

This language is implemented in an online tool called the LegAi editor, which is briefly introduced.

The work done in this paper aims at getting legal experts involved in the creation of legal knowledge. Towards this goal, we currently collaborate with a German GDPR consultancy firm. First impressions show that the creation of legal knowledge by a non-technical legal expert is possible via the annotations editor and wizard. They also confirm that a comparison tool can help the validation of the formal interpretations and therefore raise the possibility of accountability.

Our direct future work is to implement a set of computational tools which can take advantage of the created formal interpretations. We are currently developing a tool which can help the different consultants of the firm attain uniformity of legal advice by using the tool to guide them in the consultancy process.

References

[1] A.-M. Burley, W. Mattli, Europe before the court: A political theory of legal integration, International organization 47 (1993) 41–76.

[2] D. A. Weaver, B. Bimber, Finding news stories: a comparison of searches using lexisnexis and google news, Journalism & Mass Communication Quarterly 85 (2008) 515–530.

[3] M. Hashmi, G. Governatori, Norms modeling constructs of business process compliance management frameworks: a conceptual evaluation, Artificial Intelligence and Law 26 (2018) 251–305.

[4] M. Palmirani, G. Governatori, Modelling legal knowledge for gdpr compliance checking., in: JURIX, volume 313, 2018, pp. 101–110.

[5] T. Libal, M. Pascucci, Automated reasoning in normative detachment structures with ideal conditions, in: Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law, 2019, pp. 63–72.

[6] H. Prakken, G. Sartor, Law and logic: A review from an argumentation perspective, Artificial intelligence 227 (2015) 214–245.

[7] D. A. Waterman, A guide to expert systems, Addison-Wesley Longman Publishing Co., Inc., 1985.

[8] N. Connell, Expert systems in accountancy: a review of some recent applications, Accounting and Business Research 17 (1987) 221–233.

[9] J. Durkin, Expert systems: a view of the field, IEEE Intelligent Systems 11 (1996) 56–63.
[10] K. L. McGraw, K. Harbison-Briggs, Knowledge acquisition: Principles and guidelines, Prentice-Hall, Inc., 1989.
[11] M. J. Sergot, F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, H. T. Cory, The british nationality act as a logic program, Communications of the ACM 29 (1986) 370–386.
[12] P. Leith, Fundamental errors in legal logic programming, The Computer Journal 29 (1986) 545–552.
[13] M. Mills, Artificial intelligence in law: The state of play 2016, Thomson Reuters Legal executive Institute (2016).
[14] C. Langley, C. Spenser, Visirule tutorial, Copyright (c) (2007).
[15] D. Nute, Defeasible deontic logic, volume 263, Springer Science & Business Media, 2012.
[16] T. Nipkow, M. Wenzel, L. C. Paulson, Isabelle/HOL: a proof assistant for higher-order logic, Springer, 2002.
[17] T. Novotná, T. Libal, An evaluation of methodologies for legal formalization, in: International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems, Springer, 2022, pp. 189–203.
[18] M. Soavi, N. Zeni, J. Mylopoulos, L. Mich, From legal contracts to formal specifications: A systematic literature review, SN Computer Science 3 (2022) 1–25.
[19] R. Kowalski, Logical english, Proceedings of Logic and Practice of Programming (LPOP) (2020).
[20] L. Robaldo, C. Bartolini, M. Palmirani, A. Rossi, M. Martoni, G. Lenzini, Formalizing gdpr provisions in reified i/o logic: the dapreco knowledge base, Journal of Logic, Language and Information 29 (2020) 401–449.
[21] T. Libal, Towards automated gdpr compliance checking, in: International Workshop on the Foundations of Trustworthy AI Integrating Learning, Optimization and Reasoning, Springer, 2020, pp. 3–19.
[22] M. Mockus, M. Palmirani, Legal ontology for open government data mashups, in: 2017 Conference for E-Democracy and Open Government (CeDEM), IEEE, 2017, pp. 113–124.
[23] C. Bartolini, G. Lenzini, C. Santos, An interdisciplinary methodology to validate formal representations of legal text applied to the gdpr (2018).
[24] C. Benzmüller, Universal (meta-) logical reasoning: Recent successes, Science of Computer Programming 172 (2019) 48–62.
[25] T. Libal, A meta-level annotation language for legal texts, in: International Conference on Logic and Argumentation, Springer, 2020, pp. 131–150.
[26] A. Abidi, T. Libal, A validation process for a legal formalization method, in: Workshop on Methodologies for Translating Legal Norms into Formal Representations, 2022.
[27] T. Routen, T. Bench-Capon, Hierarchical formalizations, International Journal of Man-Machine Studies 35 (1991) 69–93.
[28] B. C. Pierce, Types and programming languages, MIT press, 2002.

[29] A. Church, A formulation of the simple theory of types, The journal of symbolic logic 5 (1940) 56–68.
[30] A. Steen, C. Benzmüller, The higher-order prover leo-iii, in: International Joint Conference on Automated Reasoning, Springer, 2018, pp. 108–116.
[31] K. L. Clark, Negation as failure, in: Logic and data bases, Springer, 1978, pp. 293–322.
[32] A. Colmerauer, An introduction to prolog iii, in: Computational Logic, Springer, 1990, pp. 37–79.
[33] J. Hughes, The design of a pretty-printing library, in: International School on Advanced Functional Programming, Springer, 1995, pp. 53–96.