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

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

[1] Legal Innovation Lab Wales, Swansea University livio.robaldo@swansea.ac.uk
[2] Huddersfield University {s.batsakis,i.tachmazidis}@hud.ac.uk
[3] University of Bologna {roberta.calegari,g.pisano}@unibo.it
[4] University of Calabria {francesco.calimeri,maria.morelli}@unical.it
[5] National Institute of Informatics of Japan {kuma,ksatoh}@nii.ac.jp
[6] Independent researcher guido@governatori.net

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

Keywords: Compliance checking · Normative reasoning · LegalTech

1 Introduction

LegalTech is experiencing growth in activity. Current LegalTech technologies mostly use Natural Language Processing (NLP) [17] or Machine Learning (ML) [7] to process documents and replicate legal decision-making.

However, ML is based on *statistical reasoning*: new cases are classified by similarity with the cases included in the training set. As a result, performance are intrinsically limited. Furthermore, and most important of all, as it is well-known ML tends to behave like a "black box" unable to explain its decisions and

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

it can therefore lead to biases and other discriminatory outcomes: ML trained on biased datasets tend to replicate the same biases on new inputs.

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

Logical formalization of norms requires *deontic operators* to represent the involved modalities (obligatory, permitted, prohibited) and *non-monotonic operators* fit to handle the central role of defeasibility in normative reasoning [14].

Formalizations found in past relevant literature are typically propositional, i.e. their basic components are whole propositions. However, propositions are of little usefulness for legal reasoning tasks needed within real-world LegalTech applications [1], due to their very limited expressivity. It is necessary to enhance the expressivity of the underlying logical format to the first-order level, fit to distinguish individuals from predicates and to allow the evaluation of deontic formulae to iterate over (large) sets of individuals.

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

In this paper, we follow [19], which distinguishes between monotonic knowledge, encoded within an OWL ontology for the GDPR called PrOnto [20], and non-monotonic knowledge, i.e., the deontic and defeasible legal rules that implement the selected GDPR norms, encoded within a *separate* knowledge base in LegalRuleML [2]. Following [20], in this paper we will formalize the monotonic knowledge of our use case in OWL and we will define separate legal rules in the formats that we will compare.

2 The use case

In this paper, we use the following use case:

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

The use case in (1) is a simplification of use case 2 from [4]. We simplified the use case by removing all temporal information [26]. For instance, in the original version of Article 2 the licensee is obliged to remove the material *within 24 hours* after he had published it. We believe that adding time management will not constitute a relevant additional element of comparison; although we consider it as part of our future work, it is not in the scope of the present one. Thus, we interpret norms with respect to the state of affairs holding at the time "now". If "now" the licensee has published the material without the approval *and* he has "now" removed it, then he is "now" complying with Article 2.

According to standard legal theory [29], norms are formalized as if-then rules having a deontic statement (i.e., obligation, permission, or prohibition) in the consequent and, in the antecedent, the conditions for this statement to hold true.

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

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

3 Formalizing norms at the first-order level

In this research work, we implemented the norms in (1) in six available formats for legal reasoning: SHACL [28], ASP-Core-2 [11], PROLEG [30], DLV [9], Arg2P [10], and SPINdle [16]. The if-then rules in (1) has been implemented at the first-order level, except in SPINdle in which the rules must be grounded.

Space constraints forbid us to report all formalizations in the paper. Thus, we will focus only on the two reasoners that have been identified as the "extremes" of the current state of the art: ASP-Core-2 and Arg2P. The reader is however invited to examine and execute all formalizations available on GitHub[7].

In line with [20], we stored all the monotonic knowledge of the use case within an OWL ontology, shown in Figure 1. The if-then rules representing the norms are separately formalized in the considered formats. These rules will all involve predicates corresponding 1:1 to the OWL resources in the ontology.

[7] `https://github.com/liviorobaldo/compliancecheckers`

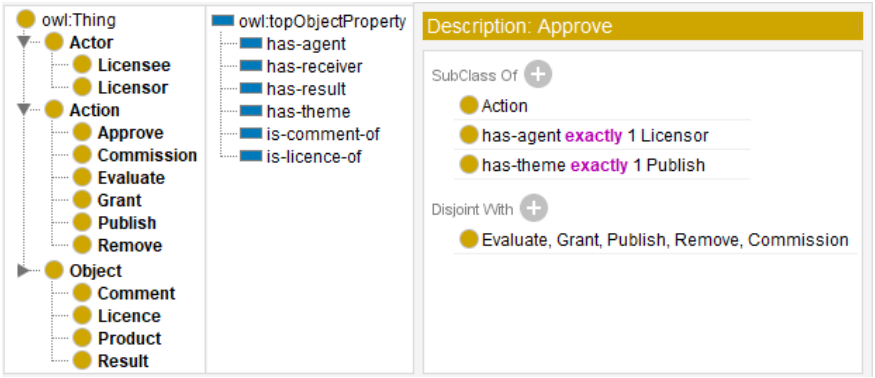

Fig. 1. Classes and object properties of the reference ontology (implemented in Protégé); the class `Approve` is shown in full detail.

Nevertheless, the concepts in Figure 1 are not enough to formalize the norms. We also need concepts to model the deontic modalities, the defeasible rules, and the compensations. These concepts are inserted in a new separate ontology shown in Figure 2. To facilitate comprehension of the formulae, in Figure 2 we introduce subclasses of `Exception` whose names directly refer to the articles in the use case denoting the exceptions. The ontology includes a further object property `compensate`, not shown in Figure 2, that relates individuals of the class `Obligation` with individuals of the (union) class `Obligatory` $\cup$ `Prohibited`. Finally, we insert a further class `Violation` whose individuals will refer to the violated (and not compensated) obligations or prohibitions.

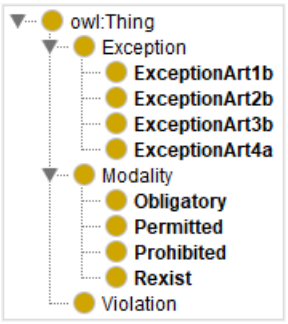

Fig. 2. Extra classes to implement deontic modalities and defeasibility

3.1 Implementing the use case in ASP

In this subsection, we formalize the if-then rules as Answer Set Programming (ASP) rules. ASP is a widely used formalism for knowledge representation and

reasoning; see [8] for an introduction. ASP is one of the most popular formalisms for AI, even at the industrial level [23]. Over the decades research has led to the definition of a variety of ASP "dialects", supported by corresponding ASP reasoners. The scientific community recently agreed on the definition of a standard input language for ASP systems, namely ASP-Core-2 [11].

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

The ASP rule encoding Art.1 in (1), which states that the licensee is prohibited to evaluate the Product unless `exceptionArt1b` holds, is shown in (2)[8].

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

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

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

exceptionArt1b(Ev) :- condition1(Ev).

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

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

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

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

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

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

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

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

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

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

(6)
```
obligatory(ca(Ep,X,R)) :- rexist(Ep), condition2(Ep,X,R).

remove(ca(Ep,X,R)) :- rexist(Ep), condition2(Ep,X,R).

hasAgent(ca(Ep,X,R),X) :- rexist(Ep), condition2(Ep,X,R).

hasTheme(ca(Ep,X,R),R) :- rexist(Ep), condition2(Ep,X,R).

compensate(ca(Ep,X,R),Ep) :- rexist(Ep), condition2(Ep,X,R).
```

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

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

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

exceptionArt3b(Ep) :- condition4(Ep).

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

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

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

exceptionArt4a(Ep) :- condition5(Ep).

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

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

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

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

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

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

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

3.2 Implementing the use case in Arg2P

Several modern approaches to legal reasoning are based on structured argumentation [22]. These approaches provide an extra layer to the representation of the inferences by including therein the graph of the arguments that either support or reject the conclusions. Although argumentation offers more functionalities than what we need to model our use case, we still decided to consider it in our analysis given the prominent role that it is increasingly assuming in LegalTech.

In this paper we consider Arg2P [6], a lightweight Prolog-based implementation for structured argumentation in compliance with the micro-intelligence definition [10]. The research in Arg2P aims to identifying different functionalities offered by available defeasible reasoners and to allow the users to configure Arg2P on the ones they need in their domain and for the purposes of their projects.

Arg2P format allows to encode labelled defeasible inference rules each from a conjunction of premises to a conclusion. Overriding among rules is achieved

via superiority relations. Arg2P format also includes modal operators[9], "o" and "p", respectively stating whether an action is obligatory or permitted.

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

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

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

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

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

The rule implementing the obligations from Article 4 is then:

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

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

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

(14)
```
art2cPart1: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
            => o(remove(ca(Ep,X,R))).
art2cPart2: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
            => remove(ca(Ep,X,R)).
art2cPart3: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
            => hasTheme(ca(Ep,X,R),R).
art2cPart4: condition2(Ep,X,R), o(-publish(Ep)), rexist(Ep)
```

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

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

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

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

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

sup(art3b, art3a).
```

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

(16)
```
ccRuleEv: o(-evaluate(Ev)), rexist(Ev), ∼(compensated(Ev))
          => violation(viol(Ev)).

ccRuleEp1: o(-publish(Ep)), rexist(Ep), ∼(compensated(Ep))
           => violation(viol(Ep)).
```

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

(17)
```
ccRuleEp2: o(publish(Ep)), ∼(rexist(Ep)), ∼(compensated(Ep))
           => violation(viol(Ep)).

ccRuleEr: o(remove(Er)), ∼(rexist(Er)), ∼(compensated(Er))
          => violation(viol(Er)).
```

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

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

ccRuleComp2: compensate(Y,X), rexist(Y) => compensated(X).
```

4 Comparison, discussion, and future works

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

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

Table 1. Time performance of the compliance checkers

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

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

Since Arg2P is one of the most modern implemented reasoners for structured argumentation, the assessed slow performance definitely demands for much further research. Structured argumentation has been mainly studied so far from a theoretical point of view but it is time now to research ways of making the theoretical findings usable in practice. This could be perhaps achieved by modeling problems in argumentation precisely as problems in ASP, in order to make the most of the format's efficiency, a solution already advocated in [8].

Similar considerations hold for PROLEG. However, contrary to Arg2P, PROLEG is not a stand-alone legal reasoner. It is a library that must be loaded within other Prolog reasoners, e.g., SWI Prolog[10]. Thus, carrying out further research to improve PROLEG efficiency most likely amounts to carrying out further research to improve the efficiency of reasoners for standard Prolog.

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

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

Unfortunately, although ASP is so efficient, achieving explainability in ASP could be difficult because, as explained in subsection 3.1 above, ASP is a declara-

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

tive language in which the reasoner tries to satisfy all rules *at once*. The returned answer set does not specify which rules have been applied to obtain the facts within the answer set. This knowledge must be inferred through an additional "reverse engineering" process, from the returned answer set to the asserted rules.

Achieving explainability in ASP is a matter of ongoing research [13]. Several techniques and methodologies to debug answer-set programs have been proposed, among which [18] and [12]. The common insight of these solutions is to add an extra-layer that relates the facts in the returned answer set with the rules that derive them, thus allowing to trace the inferential process.

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

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

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

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

Moreover, we observe that allowing the Arg2P operator "`o`" to also accept a conjunction of predicates does not appear to solve the problem so simply. In Standard Deontic Logic (SDL)[11], which inspired the definition of these operators, the axiom $\mathtt{o}(P_1, P_2) \rightarrow (\mathtt{o}(P_1), \mathtt{o}(P_2))$ holds. Thus, we may derive, for instance:

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

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

Also the modal operators of the LegalRuleML standard[12] suffer from the same problem. Possible solutions within future versions of the standard are still under discussion within the LegalRuleML technical committee (TC)[13].

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

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

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

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

5 LegalRuleML

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

This is true also in AI for the legal domain, as witnessed by lot of recent literature, e.g., [32] and [5], as well as related initiatives such as the yearly EXplainable & Responsible AI in Law (XAILA) workshop[14].

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

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

In the future, we intend to define such a methodology around LegalRuleML, which became an OASIS standard very recently, i.e., on August 30th, 2021[15].

LegalRuleML is an XML-based semi-formal language aiming at enhancing the interplay between experts in law and experts in logic. By "semi-formal" we intend that no formal model-theoretic semantics is associated with LegalRuleML. Well-formed LegalRuleML representations need to be translated into another language having such a semantics, e.g., the input formats of the legal reasoners considered above, similarly to what is done in Reaction RuleML 1.0 via the so-called "semantics profiles" [21].

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

The advocated annotations in LegalRuleML may be facilitated via a special graphical editor that allows to compose the if-then rules and store them in the XML standard. Something similar has been recently done in [25], which present a graphical editor to annotate norms in reified I/O logic [27], a novel deontic logic based on reification [15] [24] which features a computational complexity lower than standard approaches based on possible-world semantics [31]. The

[14] `https://www.geist.re/xaila:start`
[15] See `https://www.oasis-open.org/standard/legalruleml`

LegalRuleML annotations so produced can be then automatically translated in a computational language to check the compliance of the denoted norms with respect to a given state of affairs.

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

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

6 Conclusions

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

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

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

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

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

References

1. Antoniou, G., Baryannis, G., Batsakis, S., Governatori, G., Islam, M.B., Liu, Q., Robaldo, L., Siragusa, G., Tachmazidis, I.: Large-scale legal reasoning with rules and databases. Journal of Applied Logics - IfCoLog Journal **8**(4), 911–940 (2021)
2. Athan, T., Governatori, G., Palmirani, M., Paschke, A., Wyner, A.: LegalRuleML: Design Principles and Foundations. Springer International Publishing (2015)
3. Bartolini, C., Giurgiu, A., Lenzini, G., Robaldo, L.: Towards legal compliance by correlating standards and laws with a semi-automated methodology. In: BNCAI. Communications in Computer and Information Science, vol. 765. Springer (2016)

4. Batsakis, S., Baryannis, G., Governatori, G., Tachmazidis, I., Antoniou, G.: Legal representation and reasoning in practice: A critical comparison. In: Legal Knowledge and Information Systems, JURIX (2018)
5. Bibal, A., Lognoul, M., de Streel, A., Frénay, B.: Legal requirements on explainability in machine learning. Artificial Intelligence and Law **29**(2), 149–169 (2021)
6. Billi, M., Calegari, R., Contissa, G., Lagioia, F., Pisano, G., Sartor, G., Sartor, G.: Argumentation and defeasible reasoning in the law. J **4**(4) (2021)
7. Boella, G., Di Caro, L., Rispoli, D., Robaldo, L.: A system for classifying multi-label text into eurovoc. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law. pp. 239–240. ICAIL '13, ACM (2013)
8. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
9. Buccafurri, F., Faber, W., Leone, N.: Disjunctive logic programs with inheritance. Theory and Practice of Logic Programming **2** (2002)
10. Calegari, R., Omicini, A., Pisano, G., Sartor, G.: Arg2P: an argumentation framework for explainable intelligent systems. J. of Logic and Computation **32** (2022)
11. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., Schaub, T.: Asp-core-2 input language format. Theory and Practice of Logic Programming **20**(2), 294–309 (2020)
12. Cuteri, B., Dodaro, C., Ricca, F.: Debugging of answer set programs using paracoherent reasoning. In: Casagrande, A., Omodeo, E.G. (eds.) Proc. of the 34th Italian Conference on Computational Logic (CILC 2019). CEUR Workshop Proceedings, vol. 2396. CEUR-WS.org (2019)
13. Dauphin, J., Satoh, K.: Explainable ASP. In: Baldoni, M., Dastani, M., Liao, B., Sakurai, Y., Zalila-Wenkstern, R. (eds.) Proc. of 22nd international conference on Principles and Practice of Multi-Agent Systems (PRIMA 2019). Lecture Notes in Computer Science, vol. 11873. Springer (2019)
14. Gabbay, D., Horty, J., Parent, X., van der Meyden, R., van der Torre, L.: Handbook of Deontic Logic and Normative Systems. College Publications (2013)
15. Hobbs, J.R.: Deep lexical semantics. In: Proc. of the 9th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2008). Haifa, Israel (2008)
16. Lam, H.P., Governatori, G.: The Making of SPINdle. In: Proc. of International Symposium on Rule Interchange and Applications (RuleML) (2009)
17. Nanda, R., Di Caro, L., Boella, G., Konstantinov, H., Tyankov, T., Traykov, D., Hristov, H., Costamagna, F., Humphreys, L., Robaldo, L., Romano, M.: A unifying similarity measure for automated identification of national implementations of european union directives. In: Proc. of the 16th Edition of the International Conference on Articial Intelligence and Law (ICAL 2017). Association for Computing Machinery (2017)
18. Oetsch, J., Pührer, J., Tompits, H.: Stepwise debugging of answer-set programs. Theory and Practice of Logic Programming **18**(1) (2018)
19. Palmirani, M., Governatori, G.: Modelling legal knowledge for GDPR compliance checking. In: Legal Knowledge and Information Systems (JURIX) (2018)
20. Palmirani, M., Martoni, M., Rossi, A., Bartolini, C., Robaldo, L.: Pronto: Privacy ontology for legal compliance. In: EU conference on digital government (2018)
21. Paschke, A.: Reaction ruleml 1.0 for rules, events and actions in semantic complex event processing. In: Bikakis, A., Fodor, P., Roman, D. (eds.) Rules on the Web. From Theory to Applications. Springer International Publishing (2014)
22. Prakken, H., Sartor, G.: Law and logic: A review from an argumentation perspective. Artificial Intelligence and Law **227**, 214–245 (2015)

23. Reale, K., Calimeri, F., Leone, N., Ricca, F.: Smart devices and large scale reasoning via ASP: tools and applications. In: Cheney, J., Perri, S. (eds.) Practical Aspects of Declarative Languages - 24th International Symposium, year = 2022
24. Robaldo, L.: Distributivity, collectivity, and cumulativity in terms of (in)dependence and maximality. The Journal of Logic, Language, and Information **20(2)**, 233–271 (2011)
25. Robaldo, L., Bartolini, C., Palmirani, M., Rossi, A., Martoni, M., Lenzini, G.: Formalizing gdpr provisions in reified i/o logic: the dapreco knowledge base. The Journal of Logic, Language, and Information **29** (2020)
26. Robaldo, L., Caselli, T., Russo, I., Grella, M.: From Italian text to TimeML document via dependency parsing. In: Proc. of Computational Linguistics and Intelligent Text Processing (CICLing 2011). (2011)
27. Robaldo, L., Sun, X.: Reified input/output logic: Combining input/output logic and reification to represent norms coming from existing legislation. The Journal of Logic and Computation **7** (2017)
28. Robaldo, L.: Towards compliance checking in reified I/O logic via SHACL. In: Maranhão, J., Wyner, A.Z. (eds.) Proc. of 18th International Conference for Artificial Intelligence and Law (ICAIL 2021). ACM (2021)
29. Sartor, G.: Legal concepts as inferential nodes and ontological categories. Artificial Intelligence and Law **17**(3), 217–251 (2009)
30. Satoh, K., Asai, K., Kogawa, T., Kubota, M., Nakamura, M., Nishigai, Y., Shirakawa, K., Takano, C.: PROLEG: An Implementation of the Presupposed Ultimate Fact Theory of Japanese Civil Code by PROLOG Technology. In: Onada, T., Bekki, D., McCready, E. (eds.) New Frontiers in Artificial Intelligence (2011)
31. Sun, X., Robaldo, L.: On the complexity of input/output logic. The Journal of Applied Logic **25**, 69–88 (2017)
32. Waltl, B., Vogl, R.: Explainable artificial intelligence – the new frontier in legal informatics. Jusletter IT 22 (2018)