Efficient compliance checking of RDF data

LIVIO ROBALDO, *Legal Innovation Lab Wales, Swansea University, Singleton Park, Sketty, Swansea, SA28PP, UK.*

FRANCESCO PACENZA, *Department of Mathematics and Computer Science, University of Calabria, Via Pietro Bucci, cubo 30B, Rende (CS), 87036, Italy.*

JESSICA ZANGARI, *Department of Mathematics and Computer Science, University of Calabria, Via Pietro Bucci, cubo 30B, Rende (CS), 87036, Italy.*

ROBERTA CALEGARI, *Department of Computer Science and Engineering, University of Bologna, Mura Anteo Zamboni 7, Bologna, 40126, Italy.*

FRANCESCO CALIMERI, *Department of Mathematics and Computer Science, University of Calabria, Via Pietro Bucci, cubo 30B, Rende (CS), 87036, Italy and DLVSystem SRL, Viale della Resistenza 19, Quattromiglia (CS), 87036, Italy.*

GIOVANNI SIRAGUSA, *Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, Torino, 10149, Italy.*

Abstract

Automated compliance checking, i.e. the task of automatically assessing whether states of affairs comply with normative systems, has recently received a lot of attention from the scientific community, also as a consequence of the increasing investments in Artificial Intelligence technologies for the legal domain (LegalTech). The authors of this paper deem as crucial the research and implementation of compliance checkers that can directly process data in RDF format, as nowadays more and more (big) data in this format are becoming available worldwide, across a multitude of different domains. Among the automated technologies that have been used in recent literature, to the best of our knowledge, only two of them have been evaluated with input states of affairs encoded in RDF format. This paper formalizes a selected use case in these two technologies and compares the implementations, also in terms of simulations with respect to shared synthetic datasets.

1 Introduction

LegalTech, i.e. the use of Artificial Intelligence (AI) for the legal domain, is experiencing growth in activity, also at the industrial level, due to the ever-growing amount of due diligence documents that companies must submit to the governmental authorities in order to prove their compliance with the in-force regulations.[1]

Current LegalTech solutions available in the market mostly use Natural Language Processing (NLP) to extract information from textual legal documents and Machine Learning (ML) to replicate legal decision-making, in order to assist and speed up the work of legal practitioners [43], [15].

[1] See, for instance, Thomson Reuters's report at https://www.thomsonreuters.com/en-us/posts/investigation-fraud-and-risk/cost-of-compliance-2021.

Vol. 33, No. 8, © The Author(s) 2023. Published by Oxford University Press.

Advance Access Publication on 6 June 2023 https://doi.org/10.1093/logcom/exad034

However, ML is based on *statistical reasoning*. Thus, it tends to behave like a 'black box' unable to explain its decisions. Furthermore, ML trained on biased datasets tends to replicate the same biases on new inputs, thus leading to discriminatory outcomes such as refusing a loan due to race or gender.

In order to create transparent and responsible AI systems, recent research has been devoted to *explainable* AI (XAI). Although explainability does not prevent biases per se, it provides human-understandable explanations 'to determine whether discrimination occurred' [46]. XAI is particularly relevant in LegalTech (cf. [47]), as the application of the law must be transparent and bias-free by definition.

Many approaches to XAI requires the construction of symbolic representations of norms such as semantic annotations [33], legal ontologies [31] and logical rules [40]. Symbolic representations enable human-understandable forms of *logical reasoning* in place of the statistical inferences, or possibly in combination with them [5]. The chain of logical derivations on symbols provides intelligible explanations of AI decision-making.

Nevertheless, building and keeping up-to-date symbolic representations are highly time-consuming tasks because they require a lot of manual work. To mitigate manual efforts, standardized formats should be used, in order to facilitate the funnelling of efforts from more people, as well as the sharing and reuse of the resources.

All this is well-known by the community working with Semantic Web technologies and to the World Wide Web Consortium (W3C) that, for this reason, already defined several computational standard formats to support the creation of linked data over the Web.[2] Resource Description Framework (RDF), Web Ontology Language (OWL) and the Simple Protocol And Rdf Query Language (SPARQL) are the three main W3C formats for the Semantic Web. We assume that the readers of this paper already have a general knowledge of these three formats.

The activities engendered by the adoption and dissemination of the mentioned W3C standards led in the past two decades to the creation of huge amounts of RDF datasets, as well as to the development of libraries for connecting the standards with main programming languages and efficient NoSQL database management systems for working with RDF datasets in industrial contexts. These activities are expected to continue in the forthcoming years; thus, it is reasonable to envisage that the W3C standards for the Semantic Web will likely serve as the basis of XAI based on symbolic representations.

Based on this hypothesis, the authors of this paper deem research into solutions able to directly process RDF data as crucial. Automated reasoners have been proposed and investigated for decades, but most of them have their own input format into which RDF must be converted. The computational cost of the conversion would be unsustainable in a real-world (industrial) context dealing with big data. To avoid the conversion, the automated reasoners must be able to accept as input and process RDF data.

This article is concerned with the formalization in computational logic of norms that we may find in existing legislation, fit to check their compliance with a given state of affairs. In addition, a requirement of the present work is to only consider states of affairs encoded in RDF format, in light of the hypothesis explained above.

The next section will review some past literature on automated compliance checking. The literature review will reveal that a lot of work still needs to be done in order to achieve the objective of efficiently working with RDF data.

Most proposed solutions in the literature for automating compliance checking have been investigated only in theoretical terms or they have been only supported by sample implementations

[2] https://www.w3.org/2001/sw/wiki/Main_Page

that have not been empirically tested on large datasets or they lack a direct computational comparison with their competitors, i.e. with alternative available reasoners for compliance checking.

The recent contribution in [41] was a seminal work in this sense because, to the best of our knowledge, it is the first work in which several main automated reasoners (seven in total) used in recent research for compliance checking have been compared and evaluated on synthetic datasets of increasing size.

Each reasoner investigated in [41] has been evaluated on datasets encoded *in its own input format*. For only one of them the input format is RDF: the automated reasoner originally proposed in [39], which is based on Shapes Constraint Language (SHACL) rules. SHACL, briefly described in the Section 3 below, is a recent W3C standard that has been precisely designed to validate and reason with RDF datasets.

The mentioned SHACL reasoner, however, is not the most efficient reasoner among those analysed in [41]. This is DLV2 [1], which is a Datalog and Answer Set Programming reasoner, accepting in input logic formulae encoded in the ASP-Core-2 standard [10].

Therefore, the two reasoners appear to be in a trade-off: DLV2 is faster than SHACL; however, the latter should be still preferred when processing data in RDF format because, as pointed out above, the computational cost of translating RDF into ASP-Core-2 will substantially worsen DLV2's performance and usage.

However, it must be pointed out that the above considerations about the role of the RDF format for big data encoding and processing are very well-known to the ASP scientific community. The increasing use of ASP in real-world scenarios over recent years has been continuously fostering definitions and extensions of methods and tools for successfully dealing with practical applications, up to the industrial level. There is a common agreement on the need for efficiently handling multiple queries and reasoning tasks over large-sized knowledge bases in RDF.

In light of this, the DLV2 reasoner has been extended in order to connect with both relational and graph databases via explicit directives for importing/exporting data [11]. The DLV2 reasoner will be presented in section 4 below, together with its specific directives to work with RDF datasets.

The paper will then empirically investigate, in Section 5, to what extent the directives to make RDF processable in DLV2 affect the overall performance of the reasoner. In particular, this paper wants to empirically assess whether the use of these directives makes the overall performance of DLV2 *lower* than the one of the SHACL reasoner used in [41]. In cases where it does, we will conclude that the aforementioned trade-off still holds and SHACL should be indeed preferred when processing RDF datasets. In cases where it does not, we will conclude that DLV2 is more efficient than SHACL, even when processing data in the input format of the latter.

All source codes are publicly available online at https://github.com/liviorobaldo/complianceCheckingOnRDF, together with instructions to reproduce the simulations.

2 Background: compliance checking on RDF data

The first approaches to normative reasoning on RDF triples are dated in the years 2005–2015. Examples are [23] and [14]. These approaches use RDF to model the states of affairs and *separate* knowledge bases of legal rules encoded in special XML formats such as SWRL[3] or LKIF-rules.[4] When executed by suitable legal reasoners, e.g. Carneades [22], these rules check compliance or perform other legal inferences on the states of affairs encoded in RDF.

[3] https://www.w3.org/Submission/SWRL
[4] http://www.estrellaproject.org/doc/D1.1-LKIF-Specification.pdf

In the same spirit, [17] and [30] respectively propose to use RuleML[5] and LegalRuleML [4] to this end. RuleML and LegalRuleML are two standards distributed by the Organization for the Advancement of Structured Information Standards (OASIS) consortium,[6] an authoritative and influential non-profit standards body in the world. RuleML is an XML mark-up language to encode generic logical rules, while LegalRuleML is an extension of RuleML for the legal domain, i.e. specialized for encoding *legal* rules.

The approaches in [17] and [30] share with the present one the aim of using *standardized* languages to encode legal knowledge. On the other hand, RuleML and LegalRuleML are abstract XML-based languages, i.e. they do not have a well-defined model-theoretic interpretation nor, consequently, are there automated reasoners able to compute RuleML and LegalRuleML representations.

Two solutions are possible in order to make executable RuleML and LegalRuleML representations: (1) defining so-called 'semantics profiles' to provide a model-theoretic interpretation of the XML structures or (2) translating the representations into other executable languages. [38] present and discuss possible ways to implement the solution (1) with respect to RuleML. An example of solution (2) is instead [34], which converts LegalRuleML rules into the input format of the SPINdle legal reasoner [29].

Some contemporary approaches, e.g. [9] and [18], propose to encode legal rules in OWL2 decidable profiles, in order to keep computational complexity under control. Norms are represented as OWL restrictions, which are special classes. These restrictions refer to the subsets of individuals that comply with the norms. Compliance checking is then enforced via OWL2 subsumption.

[9] and [18] are also in the direction of our overall research objective, i.e. encoding legal knowledge by using W3C or OASIS standards. Nevertheless, OWL2, being a monotone format, is not expressive enough to encode legal rules, which require non-monotonic operators fit to handle the central role of defeasibility in normative reasoning [19].

[9], §3.3, acknowledge themselves that their approach does not really involve normative reasoning, which is defeasible in nature, but it is only limited to GDPR policy validation. On the other hand, as an example of full normative reasoning on the GDPR, they indicate [34], which, as said above, use LegalRuleML and the SPINdle reasoner.

Analogously, although [18] formalize compliance checking of some selected norms in OWL2, these norms are actually simpler[7] than the ones included in the use case considered here (see Section 5 below). For future developments of their framework aiming at addressing more complex deontic statements, [18] suggest themselves using SHACL in place of OWL.

Defeasibility could be still represented in OWL by modelling exceptions and other non-monotone operators in terms of OWL resources, as proposed in [13]; however, such a route appears difficult and unpractical.

Alternatively, one could try to use SPARQL that, although it was originally proposed as a query language for RDF, in the past decades, it has been extended with plenty of operators, including non-monotonic ones, which have greatly enhanced the language's expressivity and reasoning capabilities. A preliminary approach in these lines is [20], which proposed to extend the LegalRuleML meta model [3] and then to represent normative rules via SPARQL queries.

Nevertheless, what SPARQL still lacks, in our view, is the ability to establish specific *execution orders* of the rules. SPARQL is still a querying language, not a programming one; therefore, in order

[5] http://www.ruleml.org

[6] https://www.oasis-open.org

[7] Specifically, [18] do not consider compensatory norms (see [25] and [26]), e.g. Article 2 of the use case in Section 5, nor nested permissions (see [40], §4.1), e.g. Article 3 of the use case in section 5.

to execute a set of SPARQL rules in a well-defined order, it is necessary to embed the rules within another programming language, e.g. Java or C++, in which the execution order of the rules can be decided programmatically.

This is exactly what we are going to propose below. SHACL and allow the embedding of SPARQL queries to access and work with RDF triples; however, contrary to Java and C++, which are general-purpose programming languages, SHACL and DLV2 are specialized and optimized to encode and execute logical (if-then) rules.

The next section will present SHACL while Section 4 below will present DLV2. Section 5 will also illustrate the use case and the empirical comparison of the two reasoners.

3 The Shapes Constraint Language (SHACL)

The Shapes Constraint Language (SHACL) is a W3C recommendation proposed in 2017 for the purpose of validating RDF datasets. SHACL allows users to specify special constraints, called 'SHACL shapes', on RDF resources. External validators allow for checking whether RDF datasets are valid or not with respect to a set of SHACL shapes.

Simply put, SHACL shapes roughly parallel XSD grammars with respect to XML files. Another analogy between XSD/XML and SHACL/RDF is that XSD is itself written in XML like SHACL is itself written in RDF. In other words, SHACL is formally defined as a vocabulary of RDF classes and properties that allow validation rules to be described, i.e. SHACL shapes, on other RDF graphs.

SHACL is more expressive than OWL and may be therefore used as an alternative to it. In particular, SHACL includes non-monotonic operators such as negation-as-failure. As pointed out above, these are not allowed in OWL, which is a monotone language.

Furthermore, SHACL shapes are more flexible and easier to edit than OWL constraints because, while the latter are all executed at once via available OWL reasoners such as HermiT [21], in SHACL we may decouple complex validation tasks into (simpler) sequential modules.

This is possible thanks to the introduction of SHACL rules[8] that enable non-ontological types of inference, such as collecting data from RDF resources located in different, and possibly distant, parts of the ontology or computing partial results needed for the validation [37]. In other words, each SHACL shape may be associated with a set of SHACL rules that gather and aggregate information from the RDF resources; in a subsequent step, the SHACL shapes check whether the results obtained through these rules comply or not with certain validation criteria.

Most important of all, SHACL allows *priorities* on the rules to be specified, thus allowing for the definition of sequences or even flow charts of rules. Thanks to this, SHACL may be used for a variety of purposes besides validation. For instance, [41], and the present paper propose to use SHACL rules for automated compliance checking. Other recent proposals that use SHACL for other tasks in AI are [35], [36] and [16].

There are two kinds of SHACL rules: TripleRule(s), which allow a single RDF triple to be added to the inferred ontology, and SPARQLRule(s), which embed SPARQL queries in the form `CONSTRUCT-WHERE`: for subgraphs that satisfy the `WHERE` clause, the subgraph in the corresponding `CONSTRUCT` clause, which may include more than one RDF triple, is added to the inferred ontology.

SPARQLRule(s) are very expressive because they add to the richness of the SPARQL language the possibility of establishing *execution orders* among the SPARQL queries, and so to create controlled

[8] https://www.w3.org/TR/shacl-af

sequences or flow charts of such queries. Therefore, SPARQLRule(s) add the last ingredient that, in our view, was missing in [20], as discussed above.

A very simple example of SPARQLRule is shown in (1). Further, more complex examples are shown below with respect to the use case in Section 5.

(1)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 0;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
  sh:construct """CONSTRUCT {?x rdf:type TBox:mortal.}
                  WHERE {?x rdf:type TBox:man.}"""]
```

The rule in (1) leads to infer that every man is mortal, i.e. that every individual belonging (`rdf:type`) to the RDF class ‘man’ also belongs to the class ‘mortal’.

The rule features `sh:order` equal to 0, so it will be executed as first: SHACL rules are executed from the lowest to the highest value of `sh:order`. The property `sh:prefixes` enables the namespaces used in the embedded SPARQL query to be specified; in one of the examples (1), we need the `rdf` prefix as well as the prefix of the hypothetical ontology that contains the classes ‘man’ and ‘mortal’; we simply term this prefix `TBox`.

The `WHERE` clause of the SPARQL query matches the variable `?x` with every instance of the class `TBox:man`. For each of them, the `CONSTRUCT` clause adds to the inferred ontology a new triple stating that the instance is also `rdf:type` of the class `TBox:mortal`.

As is well-known, most contemporary programming languages offer libraries to execute SHACL and SPARQL instructions. Our implemented SHACL reasoner uses the TopBraid SHACL v.1.3.2 library for Java[9]. This library must be used in conjunction with Apache Jena[10], which is one of the most popular free and open source Java frameworks to work with Semantic Web and Linked Data.

4 The DLV2 reasoner

DLV2 [1] is an AI tool for Knowledge Representation and Reasoning based on Answer Set Programming (ASP)—a declarative logic-oriented formalism, successfully used in both academic and industrial applications.

The typical usage of DLV2 consists in finding the solutions of computational problems. Computational problems are encoded by logic rules in a purely declarative way. DLV2 is then fed with such a program coupled with the facts representing an instance of the problem, and produces the intended models, called ‘answer sets’, which in turn correspond one-to-one to the solutions of the given instance of the problem.

Roughly, a rule corresponds to a logical implication: i.e. whenever the body is true, the head is inferred as true as well. Rules are in the form ‘`label:Head←Body.`’. The head can be a disjunction of atoms or an atomic formula, whereas the body is a conjunction of literals. A literal is a possibly negated atom, which, in turn, could be a proposition or a first-order predicate.

As a concrete example, let us consider the classical vertex-covering problem. Given an undirected graph `G = (V,E)`, the problem requires selecting a set `C ⊆ V` such that all edges are covered

[9] https://repo1.maven.org/maven2/org/topbraid/shacl/1.3.2
[10] https://jena.apache.org

(i.e. for every edge `(a, b)`$\in$`E`, either `a`$\in$`C` or `b`$\in$`C`). The following rules encode the problem in ASP:

```
r1:  inC(X) | outC(X) :- node(X).
r2:  :- edge(X, Y), not inC(X), not inC(Y).
```

Rule r_1 has a disjunctive head modelling two trivial possibilities: every node `X` can either be part of `C` or be outside `C`. Rule r_2 has an empty head corresponding to $\bot$: the only way to make the implication true is to ensure that the conjunction in the body is false as well. Intuitively, r_2 states that for every edge between two generic nodes `X` and `Y`, it is not admissible that both `X` and `Y` are not contained in `C`. Rules with an empty head are called strong constraints.

Known information can be modelled via a special form of rules, called facts. In this case, facts can be used to model the structure of the graph for which we want to determine a proper vertex-covering. Assuming to have three nodes, say 1, 2 and 3, and the three edges $(1, 2)$, $(1, 3)$ and $(2, 3)$, the following facts model such a structure:

```
node(1). node(2). node(3).
edge(1,2). edge(1,3). edge(2,3).
```

Intuitively, `node` and `edge` can be seen as database relations.

Given that ASP is fully declarative, the order of rules and facts is immaterial; thus, the translation in ASP of the vertex-covering problem parallels 1:1 the problem definition.

More generally, besides the basic syntax shown in this example, DLV2 recognizes ASP programs written according to the ASP-Core-2 standard syntax [10]. In addition, DLV2 is also able to parse RDF triples contained in input RDF file(s) and automatically translate them into ASP facts. In other words, DLV2 is endowed with some features to improve interoperability also outside ASP boundaries.

In particular, in this work, we use the directives allowing DLV2 to connect and interact with graph databases [11]. In more detail, facts can be imported via SPARQL queries, both from *local* DBs in RDF files and *remote* SPARQL *endpoints*. The syntax of these directives is as follows:

```
  #import_local_sparql("rdf_file","query",name,arity[,typeConv]).
#import_remote_sparql("endpoint_url","query",name,arity[,typeConv]).
```

The first parameter is different depending on the type of directive. In the case of the local import, `rdf_file` can be either a local file or a URL pointing to an RDF remote file, which is downloaded and treated as a local RDF file. Once the file is present in the local machine, the graph is built into the RAM memory.

On the other hand, in the remote import, the `endpoint_url` has to be a remote endpoint: the effort of building the DB graph is now up to the remote server; hence, remote directives are typically more convenient when dealing with large datasets. The `query` parameter is a SPARQL statement defining data to be imported. The `name` and `arity` parameters define the relation to be filled with imported data. The last parameter, `typeConv`, is optional and specifies the conversion for mapping RDF data types to ASP-Core-2. A conversion type can be: (1) `U_INT` (the value is converted to an unsigned integer); (2) `UT_INT` (the value is truncated to an unsigned integer); (3) `T_INT` (the value is truncated to an integer); (4) `UR_INT` (the value is rounded to an unsigned integer); (5) `R_INT` (the value is rounded to an integer); (6) `CONST` (the value is converted to a string without quotes); (7) `Q_CONST` (the value is converted to a string with quotes). Directives can be specified at any

point within an ASP program. After that, the parsing phase terminates, and DLV2 processes one by one the encountered directives. For each SPARQL query, the results are used to fill the specified relation, by automatically translating imported data into ASP facts.

The example shown above in (1), which represents that every man is mortal, is translated into the following DLV2 program:

(2)
```
1: #import_remote_sparql("http://...",
    "PREFIX rdf: <http://...#> PREFIX TBox: <http://...#>
     SELECT ?x WHERE {?x rdf:type TBox:man.}",
    man, 1, type:Q_CONST).
2: mortal(X) :- man(X).
```

In (2), line 1, a remote import is used to query and import the set of men from the RDF dataset: the first parameter is the endpoint URL, and the subsequently quoted string is the SPARQL query filling the relation `man/1`. Assuming that the mappings resulting from the query are `Jack` and `John`, then the facts `man(Jack)` and `man(John)` are obtained. In line 2, an ASP rule infers every man as mortal, i.e. `mortal(Jack)` and `mortal(John)`.

5 The use case

This paper compares SHACL with the DLV2 reasoner on the same use case used in [41]. However, while in [41] each reasoner has been evaluated on states of affairs encoded in its own input format, here we will evaluate the two reasoners on states of affairs encoded in the same format: RDF.

The present section illustrates the use case and its formalization in SHACL and the DLV2 input format. Section 6 below reports the evaluation of synthetic RDF datasets of increasing size. The use case used in [41] is the following:

(3) - **Article 1**. The Licensor grants the Licensee a licence to evaluate the Product.
- **Article 2**. The Licensee must not publish the results of the evaluation of the Product without the approval of the Licensor. If the Licensee publishes the results of the evaluation of the Product without approval from the Licensor, the material must be removed.
- **Article 3**. The Licensee must not publish comments about the evaluation of the Product unless the Licensee is permitted to publish the results of the evaluation.
- **Article 4**. If the Licensee is commissioned to perform an independent evaluation of the Product, then the Licensee has the obligation to publish the evaluation results.

According to standard legal theory [44], formalizing the articles in (3) requires identifying the several norms they denote and formalizing these norms as if-then rules having a deontic statement (i.e. an obligation, a permission or a prohibition) in the consequent and, in the antecedent, the conditions for this statement to hold true. We will then encode the if-then rules in SHACL and ASP-Core-2.

Furthermore, norms may be defeasible, in the sense that other norms may *override* them. Therefore, in order to properly formalize the articles in (3), we must also identify which if-then rules override which other ones. Overriding will be then encoded in SHACL and ASP-Core-2 by using the non-monotonic operators of the two languages, specifically negation-as-failure, which both languages implement.

Finally, some of the rules may *compensate* violation of others. These rules indeed specify obligations that, when fulfilled, repair the non-compliance of other rules. For instance, Article 2 of

the use case specifies that if the Licensee publishes the result of the evaluation without the Licensor's approval, a new obligation holds for him: the Licensee is obliged to remove them. In cases where this obligation is fulfilled, the violation had still taken place, but it has been repaired/compensated.

Compensatory norms have been scarcely investigated in the literature. To the best of our knowledge, only [25] and [26] offer formalizations of this type of norms. The formalization in [25] has been implemented within the RuleRS system [28] and the Regorous system [24], which are both based on the aforementioned SPINdle reasoner. However, the implementations of the two systems are not publicly available as they are both protected by Data61 copyright.[11]

Formalizing compensations requires the introduction of new individuals that refer to compensatory obligations. In SHACL, this may be done by creating anonymous RDF individuals via the `CONSTRUCT` clause of the embedded SPARQL query. On the other hand, ASP-Core-2 allows functional symbols to be created that refer to the (Skolemized) new individuals. The SHACL and ASP-Core-2 rules representing the meaning of the compensation from Article 2 will be respectively shown in Subsections 5.1 and 5.2 below.

Finally, it must be noticed that the identification of the norms actually depends on the *interpretation* of the natural language utterances. It is well-known that natural language is ambiguous, thus the articles in (3) could be indeed interpreted in several ways.

For instance, Article 2 states that in cases where the Licensee does not have the approval of the Licensor, he is prohibited to publish the results. On the other hand, Article 4 states that in cases where the Licensee is commissioned to perform an independent evaluation of the Product, then he is obliged to publish the results. The problem is what the Licensee must do when both preconditions holds, i.e. when he did not get the approval of the Licensor *and* he was commissioned to perform an independent evaluation. No article in (3) states what to do in such a case.

It is reasonable to think that, in such a case, the Licensee must indeed publish the results, i.e. that the independent evaluation is somehow 'stronger' than the Licensor's disapproval. However, this is actually an *interpretation* of the norms in (3), i.e. the norms leave a blank with respect to this particular context.

Blanks in existing legislation are actually rather frequent. These could lead to disputes so that the decisions on how to interpret norms in particular contexts are eventually left to judges in courts. LegalTech technologies such as the one researched here can of course mitigate the problem by automatically detecting these blanks before disputes occur. However, this requires a lot of further work beyond the scope of the present paper.

Therefore, this paper simply assumes that the (ambiguous) norms in (3) correspond to the (non-ambiguous) interpretations in (4). In a real LegalTech document management system, e.g. [7], [6], the task of obtaining (4) from (3) would be up to an NLP module in charge of detecting and possibly solving the ambiguities in the initial text (cf. [8] and [32], [27], among others).

(4) - **Article 1**.
  (a) The Licensee is prohibited to evaluate the Product.
  (b) If the Licensor grants the Licensee a licence to evaluate the Product, then the Licensee is permitted to evaluate the Product.
  (c) Article 1b overrides Article 1a.
- **Article 2**.
  (a) If the Product has been evaluated, then the Licensee is prohibited to publish the results of the evaluation of the Product.

[11] https://research.csiro.au/data61/regorous; https://research.csiro.au/bpli/tools/rulers

(b) If the Licensor approves the publishing of the results of the evaluation of the Product, then the Licensee is permitted to publish the results of the evaluation of the Product.
(c) If the Licensee publishes the results of the evaluation of the Product without approval of the Licensor, then the Licensee is obliged to remove the results of the evaluation of the Product.
(d) Article 2b overrides Article 2a.
(e) Article 2c compensates Article 2a.

- **Article 3**.
  (a) If the Product has been evaluated, then the Licensee is prohibited to publish comments about the evaluation of the Product.
  (b) If the Licensee is permitted to publish the results of the evaluation of the Product, then the Licensee is permitted to publish comments about the evaluation of the Product.
  (c) Article 3b overrides Article 3a.
- **Article 4**.
  (a) If the Licensee is commissioned to perform an independent evaluation of the Product, then the Licensee is obliged to publish the results of the evaluation of the Product.
  (b) Article 4a overrides Article 2a.

5.1 Formalizing the use case in SHACL

This section shows the formalization of the norms in the use case, interpreted as in (4), in terms of SPARQLRule(s) in SHACL. As in the sample SPARQLRule in (1), the rules in this section will use a prefix `TBox` to refer to all RDF resources that are not part of the standard. To execute the rules on the states of affairs, these will need to be encoded in terms of the same resources with prefix `TBox`.

The TBox contains a class for each action of the use case, together with object properties describing their thematic roles. The classes corresponding to the actions are `Approve`, `Commission`, `Evaluate`, `Grant`, `Publish` and `Remove`. Each individual in these classes may also belong to one of the classes representing the four considered modalities: `Obligatory`, `Permitted`, `Prohibited` and `Rexist`. Representing modalities as RDF classes, which correspond to first-order predicates, allows the representation of deontic statements as if-then rules without modal operators, which in turn permits the inferences to be carried out more efficiently (cf. [45]).

`Rexist` is imported from [42]. Individual actions belonging to the class `Rexist` are those that really exist in the state of affairs. In other words, `Rexist` is used to mark actions that really take place in the state of affairs, in light of the assumption that those actions are not ‘true’ or ‘false’; actions can really exist or not, thus we need an explicit modality to encode their real existence in the state of affairs.

In this setting, we then obtain a violation either when an action belongs to both `Prohibited` and `Rexist`, i.e. when it takes place in the state of affairs although it should not, or when it belongs to both `Obligatory` but it does not belong to `Rexist`, i.e. when it does not take place in the state of affairs although it should.

Other classes represent the arguments of the thematic roles, i.e. `Licensee`, `Licensor`, `Product`, etc. or the exceptions triggered by some of the articles in (4). These will be directly illustrated alongside the SPARQLRule(s) reported below.

Having said that, the SPARQLRule corresponding to Article 1(a) in (4), stating that the Licensee is prohibited to evaluate the Product, is shown in (5).

(5)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 1;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
  sh:construct """
    CONSTRUCT { $this rdf:type TBox:Prohibited. }
    WHERE { $this TBox:has-agent ?x. ?x rdf:type TBox:Licensee.
            $this TBox:has-theme ?p. ?p rdf:type TBox:Product.
            NOT EXISTS{$this rdf:type TBox:ExceptionArt1b}. }"""]
```

The rule in (5) is triggered on every individual of the class `Evaluate`, i.e. on every evaluating action. This is done by a special SHACL property, called `targetClass` and not shown in (5), which associates a cluster of rules with a class. Note that the `sh:order` of the rule in (5) is equal to 1; this means that the rule is executed after all rules, associated with the same `targetClass`, which have `sh:order` equal to 0.

Each evaluating action having a `Licensee` as an agent, a `TBox:Product` as theme, and that does not belong to the class `TBox:ExceptionArt1b`, is asserted by the rule in (5) as an individual of the RDF class `TBox:Prohibited`.

`NOT-EXISTS` is the SPARQL clause to implement negation-as-failure. It is true if the triple as its argument does not occur in the RDF graph. However, since RDF makes the open-world assumption, the fact that the triple does not occur in the RDF graph does not logically entail that its corresponding assertion is false in the state of affairs: the truth value of this assertion could be instead *unknown*. In other words, the fact that the evaluating action does not belong to the class `TBox:ExceptionArt1b` holds *by default*.

The rule corresponding to Article 1b and that asserts evaluating actions as individuals of the class `TBox:ExceptionArt1b` is shown in (6). This rule must, of course, be executed before the previous one because its role is one of overriding the prohibition asserted by the latter. For this reason, the rule in (6) has `sh:order` equal to 0.

The rule in (6) asserts an evaluating action as permitted, and therefore as an exception of the prohibition asserted by the rule in (5), if the theme of the evaluating action is a `TBox:Product` with `TBox:Licence`, and this licence is, in turn, the theme of a `TBox:Grant` action that really took place in the state of affairs, i.e. that also belongs to the class `TBox:Rexist`, and such that its receiver is equal to the evaluating action's agent.

(6)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 0;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
  sh:construct """
    CONSTRUCT { $this rdf:type TBox:Permitted.
                $this rdf:type TBox:ExceptionArt1b. }
    WHERE { $this TBox:has-agent ?x. ?x rdf:type TBox:Licensee.
            $this TBox:has-theme ?p. ?p rdf:type TBox:Product.
            ?l TBox:is-licence-of ?p. ?l rdf:type TBox:Licence.
            ?eg TBox:has-agent ?y. ?y rdf:type TBox:Licensor.
            ?eg TBox:has-theme ?l. ?eg rdf:type TBox:Grant.
            ?eg rdf:type TBox:Rexist. ?eg TBox:has-receiver ?x.
}"""];
```

Article 2a is represented with the rule in (7). Contrary to the previous two rules, the one in (7), as well as the rules encoding Article 2b and Article 4a, is associated with every publishing action via the SHACL property `targetClass`.

(7)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 2;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
  sh:construct """
    CONSTRUCT { $this rdf:type TBox:Prohibited. }
    WHERE { $this TBox:has-agent ?x.
            ?x rdf:type TBox:Licensee. $this TBox:has-theme ?r.
            ?r rdf:type TBox:Result. ?ev TBox:has-result ?r.
            ?ev rdf:type TBox:Evaluate. ?ev rdf:type TBox:Rexist.
            NOT EXISTS{$this rdf:type TBox:ExceptionArt2b}.
            NOT EXISTS{$this rdf:type TBox:ExceptionArt4a}. }"""]
```

The rule in (7) asserts an action of publishing as prohibited in cases where its theme is the `TBox:Result` of an action `TBox:Evaluate` that also belongs to the class `TBox:Rexist` but not to `TBox:ExceptionArt2b` or `TBox:ExceptionArt4a`. Again, the non-inclusion within the two classes representing the exceptions is enforced via negation-as-failure.

The representation of Article 2b, shown in (8), is similar to the one of Article 1b. If the action of publishing is the theme of an action `TBox:Approve` that also belongs to the class `TBox:Rexist` and whose agent is a `TBox:Licensor`, then the publishing is permitted and, consequently, it is also an `TBox:ExceptionArt2b` for Article 2a.

(8)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 0;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
  sh:construct """
    CONSTRUCT { $this rdf:type TBox:Permitted.
                $this rdf:type TBox:ExceptionArt2b. }
    WHERE { $this TBox:has-agent ?x. ?x rdf:type TBox:Licensee.
            $this TBox:has-theme ?r. ?r rdf:type TBox:Result.
            ?ev TBox:has-result ?r. ?ev rdf:type TBox:Evaluate.
            ?ev rdf:type TBox:Rexist. ?ea TBox:has-theme $this.
            ?ea rdf:type TBox:Approve. ?ea rdf:type TBox:Rexist.
            ?ea TBox:has-agent ?y. ?y rdf:type TBox:Licensor.
}"""];
```

The other exception in (7) is entailed by (9). If the action of evaluating is the theme of an action `TBox:Commission` that also belongs to the class `TBox:Rexist`, then the publishing is obligatory and, consequently, it is also an `TBox:ExceptionArt4a` for Article 2a.

(9)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 0;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
 sh:construct """
   CONSTRUCT { $this rdf:type TBox:Obligatory.
               $this rdf:type TBox:ExceptionArt4a. }
   WHERE { $this TBox:has-agent ?x. ?x rdf:type TBox:Licensee.
           $this TBox:has-theme ?r. ?r rdf:type TBox:Result.
           ?ev TBox:has-result ?r. ?ev rdf:type TBox:Evaluate.
           ?ev rdf:type TBox:Rexist. ?ec TBox:has-theme ?ev.
           ?ec rdf:type TBox:Commission. ?ec rdf:type TBox:Rexist.
}"""]
```

Article 2c, the single article in the use case that entails a compensatory obligation of a violated prohibition, is represented as in (10). The `WHERE` clause of (10) includes all triples in the `WHERE` clause of (7), i.e. the rule triggering the prohibition that the obligation entailed by (10) compensates. The `WHERE` clause of (10) contains an additional triple '`$this rdf:type TBox:Rexist`' that requires the prohibited publishing action to really exist in the state of affairs. In other words, the rule in (10) entails the compensatory obligation only if the prohibition has been *violated*.

The `CONSTRUCT` clause of (10) asserts a new anonymous RDF individual via the '`[...]`' operator. The new individual refers to the compensatory obligation, thus it is asserted as type of the class `TBox:Obligatory`. The compensatory action is an instance of `TBox:Remove` whose theme is the `TBox:Result` that should have not been published.

The rule in (10) also relates the compensatory obligation to the prohibition via the RDF property `TBox:compensate`. In cases where compensatory obligations take place in the state of affairs, i.e. in cases where they also belongs to the class `TBox:Rexist`, the associated obligations or prohibitions are compensated for.

Note that (10) must create a new individual in the ontology because the remove action is not supposed to exist within the graph when the prohibition is violated. This action is then newly created and inserted within the class/modality `TBox:Obligatory`.

(10)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 3;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
  sh:construct """
    CONSTRUCT { [rdf:type TBox:Obligatory; rdf:type TBox:Remove;
                 TBox:has-agent ?x; TBox:has-theme ?r;
                 TBox:compensate $this]. }
    WHERE { $this rdf:type TBox:Rexist. $this TBox:has-agent ?x.
            ?x rdf:type TBox:Licensee. $this TBox:has-theme ?r.
            ?r rdf:type TBox:Result. ?ev TBox:has-result ?r.
            ?ev rdf:type TBox:Evaluate. ?ev rdf:type TBox:Rexist.
            NOT EXISTS{$this rdf:type TBox:ExceptionArt2b}.
            NOT EXISTS{$this rdf:type TBox:ExceptionArt4a}. }"""];
```

Finally, the SPARQLRule representing Article 3a is shown in (11). The rule states that publishing comments of evaluations is prohibited unless `TBox:ExceptionArt3b` holds.

(11)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 2;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
  sh:construct """
    CONSTRUCT { $this rdf:type TBox:Prohibited. }
    WHERE { $this TBox:has-agent ?x.
            ?x rdf:type TBox:Licensee. $this TBox:has-theme ?c.
            ?c rdf:type TBox:Comment. ?c TBox:is-comment-of ?ev.
            ?ev rdf:type TBox:Evaluate. ?ev rdf:type TBox:Rexist.
            NOT EXISTS{$this rdf:type TBox:ExceptionArt3b}. }"""];
```

The exception is entailed by the rule in (12), which represents Article 3b. (12) states that publishing comments of evaluation is permitted if there is another publishing action that is permitted and whose theme is the result of the same evaluation.

(12)
```
sh:rule [rdf:type sh:SPARQLRule; sh:order 1;
  sh:prefixes[sh:declare
    [sh:prefix "rdf"; sh:namespace "http://..."],
    [sh:prefix "TBox"; sh:namespace "http://..."]];
  sh:construct """
    CONSTRUCT { $this rdf:type TBox:Permitted.
                $this rdf:type TBox:ExceptionArt3b. }
    WHERE { $this TBox:has-agent ?x. ?x rdf:type TBox:Licensee.
            $this TBox:has-theme ?c. ?c rdf:type TBox:Comment.
            ?c TBox:is-comment-of ?ev. ?ev rdf:type TBox:Evaluate.
            ?ev rdf:type TBox:Rexist. ?ev TBox:has-result ?r.
            ?epr TBox:has-agent ?x. ?epr TBox:has-theme ?r.
            ?epr rdf:type TBox:Publish. ?epr rdf:type
TBox:Permitted. }"""]
```

5.2 Formalizing the use case in DLV2

This subsection presents the ASP-Core-2 formulas corresponding to the SPARQLRule(s) seen in the previous subsection. The ASP-Core-2 formulas presented here will be then executed through the DLV2 reasoner.

In order to make the comparison between the two reasoners reliable, the predicates and the architecture of the ASP-Core-2 rules shown below are kept isomorphic to the RDF resources used in the above SPARQLRule(s).

The ASP-Core-2 rule encoding Article 1(a) is shown in (13). This rule, which parallels the SPARQLRule in (5), states that the Licensee is prohibited to evaluate the Product unless `exceptionArt1b` holds.

(13)
```
prohibited(Ev):-evaluate(Ev), hasAgent(Ev,X), licensee(X),
                hasTheme(Ev,P), product(P), not exceptionArt1b(Ev).
```

'`not`' is the ASP-Core-2 operator for negation-as-failure, not the one for standard negation.[12] Therefore, `not exceptionArt1b(Ev)` is true by default, i.e. it is true when `exceptionArt1b(Ev)` is either false or unknown.

The predicate `exceptionArt1b(Ev)` holds if the agent of the evaluation action is granted a licence to evaluate the product; in such a case, the evaluation is also permitted.

In SHACL, this was modelled with a single rule, shown in (6) above, which entails both the exception and the permission. On the other hand, the ASP-Core-2 syntax does not allow conjunctions of predicates in the consequent of the rules; hence, the typical approach in ASP-Core-2 is to introduce an extra predicate that refers to the condition and then uses this predicate as the antecedent of more than one rule.

Article 1(b) in (4) is then modelled via the three rules in (14), which parallel the single SPARQLRule in (6); `condition1(Ev)` is the extra predicate introduced to accommodate the syntactic limitation of ASP-Core-2 explained above.

(14)
```
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).
```

The rules in (15) represent the prohibition described in Article 2a. They parallel the SPARQLRule in (7). Note that also the rules in (15) introduce an extra predicate that refers to the antecedent of the prohibition. The reason is that this condition is also part of the antecedent of the compensatory obligation from Article 2c; see (18) below.

(15)
```
condition2(Ep, X, R) :- publish(Ep), hasAgent(Ep, X), licensee(X),
                        hasTheme(Ep, R), result(R), hasResult(Ev, R),
                        evaluate(Ev), rexist(Ev),
                        not exceptionArt2b(Ep), not exceptionArt4a(Ep).

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

The ASP-Core-2 representations of Article 2b and Article 4a, which parallel the SPARQLRule(s) in (8) and (9) above, are respectively shown in (16) and (17).

(16)
```
condition3(Ev) :- publish(Ep), hasAgent(Ep,X), hasTheme(Ep,R),
                  licensee(X), result(R), evaluate(Ev), rexist(Ev),
                  hasResult(Ev,R), hasTheme(Ea,Ep), approve(Ea),
                  rexist(Ea), hasAgent(Ea,Y), licensor(Y).

exceptionArt2b(Ep) :- condition3(Ep).

permitted(Ep) :- condition3(Ep).
```

[12] This is instead '-'.

(17)
```
condition4(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) :- condition4(Ep).

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

Both rules in (16) and (17) entail an exception for the publishing action `Ep`, in cases where `condition3(Ep)` and `condition4(Ep)` respectively hold. However, `condition3(Ep)` also entails that `Ep` is permitted while `condition4(Ep)` entails that it is obligatory.

Now, in order to model Article 2c, we must introduce a new individual that refers to the compensatory obligation denoted by the article.

The SPARQLRule that models Article 2c, shown above in (10), represents the new individual as an anonymous RDF individual added to the ABox by the `CONSTRUCT` clause.

In ASP-Core-2, we make use of functional symbols to simulate existential quantification via Skolemization. Specifically, we introduce in the formulas a function symbol 'ca' (for 'compensatory action') over the universally quantified variables `Ep`, `X` and `R`. The ASP-Core-2 rules that parallel the SPARQLRule in (10) are then:

(18)
```
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).
```

Note that the antecedents of the rules in (18) include the predicate `condition2(Ep,X,R)` defined above in (15). This predicate refers to the condition for the prohibition of publishing comments. The antecedents of the rules in (18) also require these publishing actions to really exist in the state of affairs, i.e. they require the *violation* of the prohibition.

Finally, Article 3a, which defines the prohibition of publishing comments about the evaluation of the product, is represented as in (19). The ASP-Core-2 rule in (19) parallels the SPARQLRule in (11).

(19)
```
prohibited(Ep) :- publish(Ep), hasAgent(Ep, X), licensee(X),
                  hasTheme(Ep, C), comment(C), isCommentOf(C, Ev),
                  evaluate(Ev), rexist(Ev), not exceptionArt3b(Ep).
```

The antecedent of (19) requires the negation-as-failure of `exceptionArt3b(Ep)`. This exception is entailed by (20), which parallels the SPARQLRule in (12), in cases where the licensee is permitted to publish the results of the evaluation of the product; in such a case, the publishing of the comments is also permitted.

(20)
```
condition5(Ep) :- publish(Ep), hasAgent(Ep, X), licensee(X),
                  hasTheme(Ep, C), comment(C), isCommentOf(C, Ev),
                  evaluate(Ev), rexist(Ev), hasResult(Ev, R),
                  hasTheme(Epr, R), hasAgent(Epr, X), publish(Epr),
                  permitted(Epr).

exceptionArt3b(Ep) :- condition5(Ep).

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

6 Evaluation

This section evaluates and compares the rules shown above on their respective automated reasoners. The evaluation consists in using the two automated reasoners for checking compliance of states of affairs, encoded in RDF, with respect to the rules.

States of affairs are automatically created via the dataset generator used in [41]. Each of them consists in the same set of RDF triples (in Turtle format[13]):

```
:ev rdf:type TBox:Evaluate; TBox:has-agent :x;
                  TBox:has-result :r; TBox:has-theme :p.
:epc rdf:type TBox:Publish; TBox:has-agent :x; TBox:has-theme :c.
:epr rdf:type TBox:Publish; TBox:has-agent :x; TBox:has-theme :r.
:ea rdf:type TBox:Approve; TBox:has-agent :y; TBox:has-theme :epr.
:ec rdf:type TBox:Commission; TBox:has-theme :ev.
:eg rdf:type TBox:Grant; TBox:has-agent :y;
                  TBox:has-receiver :x; TBox:has-theme :l.
:er rdf:type TBox:Remove; TBox:has-agent :x; TBox:has-theme :r.
:l rdf:type TBox:Licence; TBox:is-licence-of :p.
:c rdf:type TBox:Comment; TBox:is-comment-of :ev.
:p rdf:type TBox:Product.
:r rdf:type TBox:Result.
:x rdf:type TBox:Licensee.
:y rdf:type TBox:Licensor.
```

In which `:ev`, `:epc`, `:epr`, `:ea`, `:ec`, `:eg` and `:er` are the RDF individuals corresponding to the actions of the use case while `:x`, `:y`, `:r`, `:p`, `:c`, `:r` and `:l` are those corresponding to the actors and the objects involved in these actions.

The dataset generator creates new states of affairs by generating more indexed individuals, i.e. `:ev_1`, `:epc_1`, …, `:r_1`, `:l_1`, `:ev_2`, `:epc_2`, …, `:r_2`, `:l_2`, etc., and by duplicating the triples above while asserting them on these indexed individuals. Thus, each state of affairs is identified with the index used for generating the RDF individuals.

Finally, each state of affairs includes triples asserting which actions really exist. For each state of affairs, the dataset generator *randomly* decides which actions really took place. For instance, the dataset generator could (randomly) decide that in the state of affairs associated with the index ‘1’,

[13] https://www.w3.org/TR/turtle/

only the following four actions really exist:

```
:ev_1 rdf:type TBox:Rexist. :epr_1 rdf:type TBox:Rexist.
:ec_1 rdf:type TBox:Rexist. :eg_1 rdf:type TBox:Rexist.
```

In other words, in this state of affairs the evaluation, the publishing of the results, the commissioning and the granting of the licence really took place; on the other hand, the approval, the publishing of the comments and the removal of the results did not.

The two automated reasoners will check the compliance of the rules in this scenario. The next subsection will provide further details on how this is specifically carried out.

6.1 Checking rules' compliance on the RDF states of affairs

Further SHACL and ASP-Core-2 rules must be added for the reasoners to check compliance on the RDF datasets, i.e. to detect violations. Specifically, these additional rules will infer violations according to the following conditions:

(21) a. A violation is inferred in cases where an individual action belongs to both the extension of the predicate `Prohibited` and the extension of the predicate `Rexist`, and the condition in (21.c) does not hold.

b. A violation is inferred in cases where an individual action belongs to the extension of the predicate `Obligatory` but not to the extension of the predicate `Rexist`, and the condition in (21.c) does not hold.

c. A compensation is inferred in cases where an individual action that satisfies condition (21.a) or condition (21.b) is related, via the `compensate` binary predicate, to another action that belongs to the extension of the predicate `Rexist`. Compensations override conditions (21.a) and (21.b): compensated individual actions no longer satisfy these conditions.

(21.a) states that an action violates the normative system in cases where it is prohibited and it takes place in the state of affairs. On the other hand, (21.b) states that an action violates the normative system in cases where it is obligatory and it does not take place in the state of affairs. Finally, (21.c) states when actions are compensated, i.e. when their associated compensatory obligations take place in the state of affairs; in such a case, even if these actions entail violations, the normative system is compliant.

The rules in (21.a-c) do not change across use cases. For this reason, we plan in our future works to implement them as static built-in procedures within the automated reasoners. In other words, rather than encoding them as explicit SHACL and ASP-Core-2 rules to be executed together with the rules representing the norms, as we did in the experiments presented here (see the GitHub repository), we want to hard-code them as internal optimized routines in the programming languages in which the two reasoners are implemented, i.e. Java, for the SHACL reasoner, and C++, for DLV2.

Figure 1 shows the evaluation of the two reasoners on RDF files located in the hard disk of the PC in which the two automated reasoners have been executed. Tests have been performed on a Linux machine equipped with Intel(R) Core(TM) $i7-9750H$ CPU at 2.60GHz with 12 cores and 32GB RAM, running Ubuntu 22.04 LTS. The files have increasing size; they respectively include 50, 100, ..., 500 states of affairs encoded in RDF.

Figure 1 shows that DLV2's performance is much superior to the SHACL reasoner's.

It is also worth noticing that DLV2's processing time does not considerably vary on the input size. In particular the processing time needed by DLV2 to process the largest dataset, i.e. the one

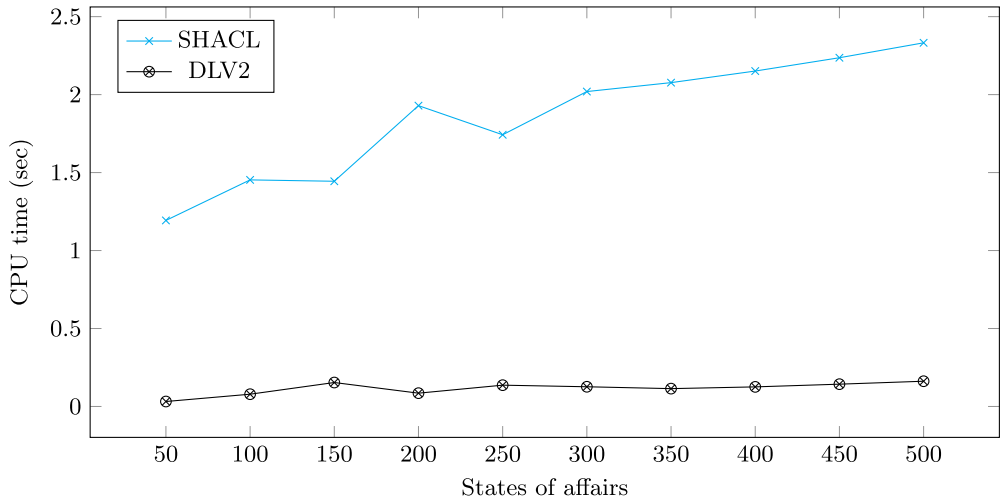

FIGURE 1. Performance of the two automated reasoners for increasing input file sizes.

containing 500 states of affairs, is only 0.008 seconds higher than the processing time needed to process the dataset including 150 states of affairs.

In our view, these empirical findings specifically show that the `#import_local_sparql` directive does not affect the overall performance of the reasoner, i.e. that importing from RDF files is computationally similar to importing from files written in ASP-Core-2 standard format, i.e. the format for which DLV2 has been originally designed.

Once the input data has been uploaded, it is not surprising that DLV2 takes almost the same time to process the datasets, regardless of their size: ASP has been primarily designed to achieve scalability on NP-hard search problems [12].

Nevertheless, as explained in the introduction, we are not actually much interested in comparing the reasoner's performance on input data imported from local files.

On the contrary, we are interested in investigating their performance in view of using them within *industrial* LegalTech applications for compliance checking. These applications are intended to process big data, i.e. large repositories including much more than 500 states of affairs [2]. It would be of course very inefficient to store these repositories within local files on hard disk. On the contrary, they should be stored within industrial DMBSs that make the RDF data available on SPARQL endpoints.

Therefore, in order to properly compare the two automated reasoners with respect to their efficiency to deal with big data, we have also made the states of affairs available on a SPARQL endpoint on localhost via Apache Jena Fuseki v4.6.1.[14] We then uploaded the states of affairs within the two reasoners by querying the endpoint via SPARQL and, finally, we checked their compliance with respect to the legal rules as before.

As explained in Section 4 above, the DLV2 reasoner offers remote import directives to contact SPARQL endpoints and retrieve RDF resources from there via SPARQL.

On the other hand, the TopBraid SHACL v.1.3.2 library, which we used to implement the SHACL automated reasoner in Java, does not offer similar directives. However, these are available in Apache

[14] https://jena.apache.org/documentation/fuseki2

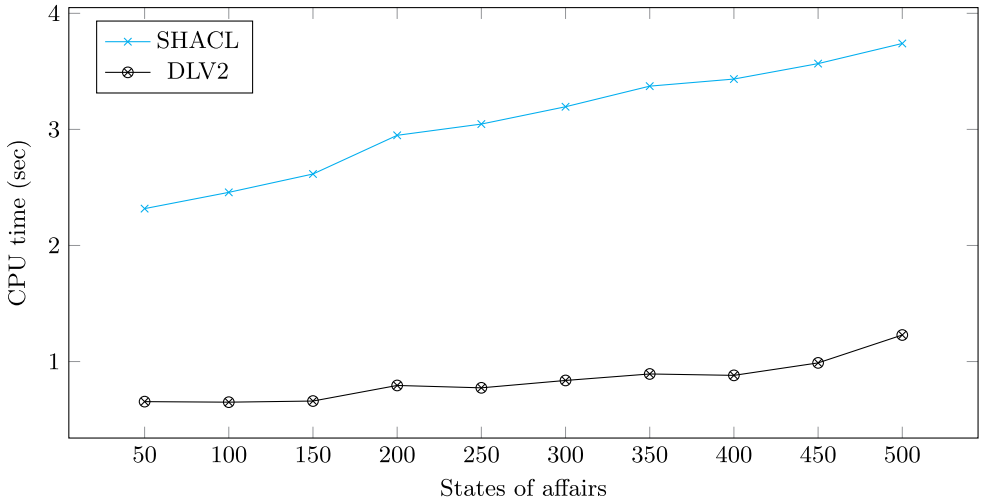

FIGURE 2. Performance of the two automated reasoners on SPARQL endpoints.

Jena. Thus, by querying the SPARQL endpoint via the routines offered by Apache Jena, we were able to rebuild the RDF models on which the SHACL rules were executed.

Figure 2 shows the performance of the reasoners with the same states of affairs used in 1 but made available via a SPARQL endpoint on localhost.

Figure 2 shows that DLV2 is much faster than the SHACL reasoner even when RDF data are imported from SPARQL endpoint.

We observe, however, that in this second experiment DLV2's performance is *not* almost unchanged depending on the size of the input dataset. Figure 2 shows instead that the DLV2's processing time does substantially increase alongside the size of the datasets.

Given the results in Figure 1, however, we believe that this is mainly due to the transfer time of the data from the server to the client, rather than to the reasoner itself. Therefore, in order to process big data stored within a SPARQL endpoint, we do not advise to transfer all states of affairs at once, but rather to create several instances of the DLV2 reasoner and execute them in parallel, with each instance processing a *portion* of the (big) dataset. However, this would require the reorganization of the ABox so that (sub)sets of states of affairs are indexed and can be thus efficiently retrieved.

7 Conclusions

This paper presented a comparison between two automated reasoners from the recent literature that are able to check compliance on states of affairs encoded in RDF format.

Big data in this format are increasingly becoming available nowadays worldwide, in a multitude of different domains. This is expected to further grow in the forthcoming years; thus, it is reasonable to envisage that the RDF format will likely serve as the basis of explainable Artificial Intelligence based on symbolic representations.

For this specific reason, we deem it crucial to research and implement compliance checkers that are able to directly process data in RDF format and we likewise hope that our work will encourage further research and developments in this direction.

In light of the above, we indeed believe that the advocated research and developments will likely have an impact on the digital economy as a whole.

It is established that well-functioning legal systems are crucial for economic performance. Since legislation is at the basis of and regulates our everyday life and societies, many categories of big data, e.g. financial data or healthcare data, must comply with, and are thus highly dependent on, the in-force norms.

Matching and annotating big data with legislative information will then produce even more and richer big data, thus the need to further investigate and develop automated technologies for the legal domain compatible and interoperable with RDF, the standard format that, in our view, will be used in the future to encode and publish data worldwide.

This paper specifically compared the reasoner based on SHACL, originally proposed in [39] and further extended in [41], with DLV2, an automated reasoner based on ASP-Core-2, available for several years and recently equipped with directives to support import of data via SPARQL [11].

The SPARQL directives of DLV2 have not previously been evaluated and compared with other reasoners on datasets of significant size; the present paper represents the first relevant attempt in this sense. The empirical results reported in Section 6 above show that DLV2 is much faster than the SHACL reasoner both for data imported from local files and for data imported from SPARQL endpoint on localhost.

Specifically, after observing the results, we do not believe that the SPARQL directives of DLV2 lower the overall performance whatsoever: the processing time for importing RDF data is basically the same for importing data in ASP-Core-2 format. It is well-known that DLV2 is high-performance for processing the data, once these are imported: the ASP format has been primarily designed to cope with NP-hard search problems.

The only reason why one should still prefer the SHACL reasoner over DLV2 is that SHACL and SPARQL are W3C standards and, consequently, plenty of open-source libraries to work with the two formats are freely available on the Web, e.g. Apache Jena. On the other hand, DLV2 is not open-source and is free only for non-commercial use[15]. A well-known open-source reasoner for ASP is Clingo[16], which currently does not directly support RDF input data and a ready-to-use SPARQL interface.

Acknowledgements

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. Francesco Calimeri, Francesco Pacenza, and Jessica Zangari acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), Spoke 9 - Green-aware AI, under the NRRP MUR program funded by the NextGenerationEU, and the support of the project PRIN PE6, Title: Declarative Reasoning over Streams", funded by the Italian Ministero dell'Università, dell'Istruzione e della Ri cerca (MIUR), CUP:H24I17000080001. The research of Roberta Calegari has been partially supported by the CompuLaw" project, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No. 833647).

[15]The commercial use of DLV2 is managed by DLVSystem (https://www.dlvsystem.it), a spin-off of the University of Calabria.

[16]https://potassco.org/clingo

[1] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri and J. Zangari. The ASP system DLV2. In *LPNMR*. Lecture Notes in Computer Science, vol. 10377, pp. 215–221. Springer, 2017.

[2] T. Athan, H. Boley, G. Governatori, M. Palmirani, A. Paschke and A. Wyner. LegalRuleML: from Metamodel to use cases. In *Theory, Practice, and Applications of Rules on the Web*, L. Morgenstern, P. Stefaneas, F. Lévy, A. Wyner and A. Paschke., eds. Springer, Berlin Heidelberg, 2013.

[3] T. Athan, G. Governatori, M. Palmirani, A. Paschke and A. Wyner. *LegalRuleML: Design Principles and Foundations*. Springer International Publishing, 2015.

[4] A. Bibal, M. Lognoul, A. De Streel and B. Frénay. Legal requirements on explainability in machine learning. *Artificial Intelligence and Law*, **29**, 149–169, 2021.

[5] G. Boella, L. Di Caro, L. Humphreys, L. Robaldo, P. Rossi and L. van der Torre. Eunomos, a legal document and knowledge management system for the web to provide relevant, reliable and up-to-date information on the law. *Artificial Intelligence and Law*, **24**, 245–283, 2016.

[6] G. Boella, L. Di Caro, L. Humphreys, L. Robaldo and L. van der Torre. NLP challenges for Eunomos a tool to build and manage legal knowledge. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012*, N. Calzolari, K. Choukri, T. Declerck, M. U. Dogan, B. Maegaard, J. Mariani, J. Odijk and S. Piperidis., eds. European Language Resources Association (ELRA), 2012.

[7] G. Boella, L. Di Caro, D. Rispoli and L. Robaldo. A system for classifying multi-label text into eurovoc. In *Proceedings of 14th International Conference for Artificial Intelligence and Law (ICAIL)*. ACM, 2013.

[8] P. A. Bonatti, L. Ioffredo, I. M. Petrova, L. Sauro and I. S. R. Siahaan. Real-time reasoning in OWL2 for GDPR compliance. *Artificial Intelligence*, **289**, 103389, 2020.

[9] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca and T. Schaub. ASP-Core-2 input language format. *Theory and Practice of Logic Programming*, **20**, 294–309, 2020.

[10] F. Calimeri, D. Fuscà, S. Perri and J. Zangari. External computations and interoperability in the new DLV grounder. In *AI*IA*. Lecture Notes in Computer Science, vol. 10640. Springer, 2017.

[11] F. Calimeri, M. Gebser, M. Maratea and F. Ricca. Design and results of the fifth answer set programming competition. *Artificial Intelligence*, **231**, 151–181, 2016.

[12] G. Casini, T. Meyer, K. Moodley, U. Sattler and I. Varzinczak. Introducing defeasibility into OWL ontologies. In *Proceedings of International Semantic Web Conference (ISWC)*, R. Meersman, P. Herrero and T. Dillon., eds, 2015.

[13] M. Ceci. Representing judicial argumentation in the semantic web. In *AI Approaches to the Complexity of Legal Systems (AICOL 2013)*. Lecture Notes in Computer Science, vol. 8929, P. Casanovas, U. Pagallo, M. Palmirani and G. Sartor., eds. Springer, 2013.

[14] I. Chalkidis and D. Kampas. Deep learning in law: early adaptation and legal word embeddings trained on large corpora. *Artificial Intelligence and Law*, **27**, 171–198, 2019.

[15] R. David, S. Ahmetaj, M. Šimkus and A. Polleres. Repairing SHACL constraint violations using answer set programming. In *Proceedings of the 21st International Semantic Web Conference (ISWC)*, 2022.

[16] M. De Vos, S. Kirrane, J. A. Padget and K. Satoh. ODRL policy modelling and compliance checking. In *Rules and Reasoning—Third International Joint Conference, RuleML+RR*, P. Fodor, M. Montali, D. Calvanese and D. Roman., eds, 2019.
[17] E. Francesconi and G. Governatori. Patterns for legal compliance checking in a decidable framework of linked open data. *Artificial Intelligence and Law* (in press), 2022.
[18] D. Gabbay, J. Horty, X. Parent, R. van der Meyden and L. van der Torre. *Handbook of Deontic Logic and Normative Systems*. College Publications, 2013.
[19] F. Gandon, G. Governatori and S. Villata. Normative requirements as linked data. In *Legal Knowledge and Information Systems*, A. Z. Wyner and G. Casini., eds, vol. 302. IOS Press, 2017.
[20] B. Glimm, I. Horrocks, B. Motik, G. Stoilos and Z. Wang. HermiT: An OWL 2 reasoner. *Journal of Automated Reasoning*, **53**, 245–269, 2014.
[21] T. F. Gordon. Combining rules and ontologies with Carneades. In *Proceedings of the 5th International RuleML2011@BRF Challenge*, S. Bragaglia, C. V. Damásio, M. Montali, A. D. Preece, C. J. Petrie, M. Proctor and U. Straccia., eds, 2011.
[22] T. F. Gordon. Constructing legal arguments with rules in the legal knowledge interchange format (LKIF). In *Computable Models of the Law*, P. Casanovas, G. Sartor, N. Casellas and R. Rubino., eds. Springer, 2008.
[23] G. Governatori. The regorous approach to process compliance. In *19th IEEE International Enterprise Distributed Object Computing Workshop*, J. Kolb, B. Weber, S. Hallé, W. Mayer, A. K. Ghose and G. Grossmann., eds. IEEE Computer Society, 2015.
[24] G. Governatori and A. Rotolo. Logic of violations: a Gentzen system for reasoning with contrary-to-duty obligations. *Australasian Journal of Logic*, **4**, 2006.
[25] G. Governatori and A. Rotolo. Time and compensation mechanisms in checking legal compliance. *Journal of Applied Logics—IfCoLog Journal*, **6**, 815–846, 2019.
[26] L. Humphreys, G. Boella, L. van der Torre, L. Robaldo, L. Di Caro, S. Ghanavati and R. Muthuri. Populating legal ontologies using semantic role labeling. *Artificial Intelligence & Law*, **29**, 171–211, 2021.
[27] M. B. Islam and G. Governatori. RuleRS: a rule-based architecture for decision support systems. *Artificial Intelligence and Law*, **26**, 315–344, 2018.
[28] H.-P. Lam and G. Governatori. The making of SPINdle. In A. Paschke, G. Governatori and J. Hall., eds, *Proceedings of International Symposium on Rule Interchange and Applications (RuleML 2009)*, Available online athttp://spindle.data61.csiro.au/spindle, 2009. Springer.
[29] H.-P. Lam, M. Hashmi and B. Scofield. In *Enabling Reasoning with LegalRuleML*, *Rule Technologies. Research, Tools, and Applications—10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6–9, 2016. Proceedings*. Lecture Notes in Computer Science, vol. 9718, J. J. Alferes, L. E. Bertossi, G. Governatori, P. Fodor and D. Roman., eds, pp. 241–257. Springer, 2016.
[30] V. Leone, L. Di Caro and S. Villata. Taking stock of legal ontologies: a feature-based comparative analysis. *Artificial Intelligence and Law*, **28**, 207–235, 2020.
[31] R. Nanda, L. Di Caro, G. Boella, H. Konstantinov, T. Tyankov, D. Traykov, H. Hristov, F. Costamagna, L. Humphreys, L. Robaldo and M. Romano. A unifying similarity measure for automated identification of national implementations of European union directives. In *Proceedings of the 16th International Conference on Artificial Intelligence and Law (ICAIL)*. ACM, 2017.

[32] A. Nazarenko, F. Lévy and A. Wyner. An annotation language for semantic search of legal sources. In *Proceedings of 11th International Conference on Language Resources and Evaluation (LREC)*. European Language Resources Association (ELRA), 2018.
[33] M. Palmirani and G. Governatori. Modelling legal knowledge for GDPR compliance checking. In *Legal Knowledge and Information Systems—JURIX*. Frontiers in Artificial Intelligence and Applications, vol. 313, M. Palmirani., ed., pp. 101–110. IOS Press, 2018.
[34] H. J. Pandit, D. O'Sullivan and D. Lewis. Exploring GDPR compliance over provenance graphs using SHACL. In *Proceedings of the Conference on Semantic Systems (SEMANTiCS 2018)*, A. Khalili and M. Koutraki., eds, 2018.
[35] P. Pareti, G. Konstantinidis, F. Mogavero and T. J. Norman. SHACL satisfiability and containment. In *Proceedings of 19th International Semantic Web Conference (ISWC)*, J. Z. Pan. et al., eds. Springer, 2020.
[36] P. Pareti, G. Konstantinidis, T. J. Norman and M. Sensoy. SHACL constraints with inference rules. In *Proceedings of 18th International Semantic Web Conference (ISWC)*, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. F. Cruz, A. Hogan, J. Song, M. Lefrançois and F. Gandon., eds. Springer, 2019.
[37] A. Paschke. Reaction RuleML 1.0 for rules, events and actions in semantic complex event processing. In *Rules on the Web. From Theory to Applications*, A. Bikakis, P. Fodor and D. Roman., eds. Springer International Publishing, 2014.
[38] L. Robaldo. Towards compliance checking in reified I/O logic via SHACL. In *Proceedings of 18th International Conference for Artificial Intelligence and Law (ICAIL)*, J. Maranhão and A. Z. Wyner., eds. ACM, 2021.
[39] L. Robaldo, C. Bartolini, M. Palmirani, A. Rossi, M. Martoni and G. Lenzini. Formalizing GDPR provisions in reified I/O logic: the DAPRECO knowledge base. *The Journal of Logic, Language, and Information*, **29**, 401–449, 2020.
[40] L. Robaldo, S. Batsakis, R. Calegari, F. Calimeri, M. Fujita, G. Governatori, M. Morelli, F. Pacenza, G. Pisano, K. Satoh, I. Tachmazidis and J. Zangari. Compliance checking on first-order knowledge with conflicting and compensatory norms—a comparison among currently available technologies. *Artificial Intelligence and Law*, in press, 2023.
[41] L. Robaldo and X. Sun. Reified input/output logic: combining input/output logic and reification to represent norms coming from existing legislation. *The Journal of Logic and Computation*, **27**, 2471–2503, 2017.
[42] L. Robaldo, S. Villata, A. Wyner and M. Grabmair. Introduction for artificial intelligence and law: special issue "natural language processing for legal texts". *Artificial Intelligence and Law*, **27**, 113–115, 2019.
[43] G. Sartor. Legal concepts as inferential nodes and ontological categories. *Artificial Intelligence and Law*, **17**, 217–251, 2009.
[44] X. Sun and L. Robaldo. On the complexity of input/output logic. *The Journal of Applied Logic*, **25**, 69–88, 2017.
[45] B. Waltl and R. Vogl. Explainable artificial intelligence—the new frontier in legal informatics. *Jusletter IT*, **22**, 2018.
[46] XAILA *Manifesto Workshop "Explainable and Responsible AI and Law"*. http://xaila.geist.re, 2022.

Received 1 May 2023