Enabling Reasoning with LegalRuleML

Ho-Pun Lam(✉), Mustafa Hashmi(✉), and Brendan Scofield

Data61, CSIRO | NICTA, Spring Hill, Australia
{brian.lam,mustafa.hashmi}@data61.csiro.au

Abstract. This paper presents an approach for the specification and implementation of translating legal norms represented using LegalRuleML to a variant of Modal Defeasible Logic. From its logical form, legal norms will be transformed into a machine readable format and eventually implemented as executable semantics that can be reasoned about depending upon the client's preference.

Keywords: Legal reasoning · LegalRuleML · Business contracts · Defeasible logic

1 Introduction

Generally regulatory rules are written in natural language— for their automated verification, they need to be transformed into a format that machines can understand. As a result, several languages such as RuleML[1], LKIF [7], SBVR [24], PENELOPE [8], ConDec language [26], ContractLog [25], OWL-S[2], have been proposed to facilitate this process. Each of these languages offer useful functionalities but is not free from shortcomings (see [9] for some of the shortcomings of these languages). For example, RuleML is an XML based prominent industry standard language for translating rules documents into a machine readable format. It provides the features that enable users to use different types of rules (such as derivation rules, fact, query, integrity constraint, etc.) to represent different kinds of elements according to their needs. However, it lacks support for the use of deontic concepts, such as obligations (such as achievement and maintenance), permission, prohibition, and is unable to handle cases with contrary-to-duty (CTD) obligations [6] that may arise from the violations of other obligations, which frequently appear in legal contracts [10].

Grosof [18] proposed to adopt courteous logic programming as execution model for RuleML rule-base for translating the clauses of a contract, which

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

[1] RuleML: The Rule Markup Initiative, http://www.ruleml.org.

[2] The OWL services coalition. OWL-S 1.2 Release, http://www.daml.org/services/owl-s/.

J.J. Alferes et al. (Eds.): RuleML 2016, LNCS 9718, pp. 241–257, 2016.
DOI: 10.1007/978-3-319-42019-6_16

filled the gap among the various types of rules in RuleML; however, their approach does not consider normative effects. Later, Governatori [10] addressed the shortcomings of [18], and extended *Defeasible Logic*(DL) with standard deontic operators for representing normative effects as well as an operator to deal with CTD obligations. This extended language also provides RuleML compliant data schemas for representing deontic elements and provides constructs to resolve some of the shortcomings that have been discussed in [9].

This paper focuses on transforming the legal norms represented using LegalRuleML into a variant of Modal Defeasible Logic [16]. As a consequence, our work reported here makes it possible to use an implementation of DL as the engine to compute the extensions on the legal norms represented using LegalRuleML and reason on it. Due to the space limit, details of other features supported by LegalRuleML, such as *Contexts* and *Alternatives*, will not be covered in this paper.

The remainder of the paper is structured as follows: in Sect. 2 we tersely discuss a short contract from [10] following which we provide some background information on DL in Sect. 3. Section 4 discusses the mapping and procedures to transform a legal theory represented using LegalRuleML to DL. Section 5 discusses related work, followed by some concluding remarks and pointers for future work.

2 A Sample Contract

In this section we discuss a sample "*Contract of Services*" based on the analysis and adapted from [10].

Contract of Services

The Deed of Agreement is entered into effects between ABC company (to be known as Purchaser) and ISP plus (to be known as Supplier) WHEREAS Purchaser desires to enter into an agreement to purchase from Supplier the application server (to be known as Goods) in this agreement. Both the parties shall enter into an agreement subject to the following terms and conditions:

1. **Definitions and Interpretations**
   1.1. All prices are in Australian current unless otherwise stated.
   1.2. This agreement is governed by the Australian law and both the parties hereby agree to submit to the jurisdiction of the Courts of the Queensland with respect to this agreement.
2. **Commencement and Completion**
   2.1. The contract enters into effects as Jan 30, 2002.
   2.2. The completion date is scheduled as Jan 30, 2003.
3. **Policy on Price**
   3.1. A *"Premium Customer"* is a customer who has spent more than $10000 in goods. Premium Customers are entitled a 5 % discount on new orders.

3.2. Goods marked as *"Special Order"* are subject to a 5 % surcharge. Premium customers are exempt from special order surcharge.
3.3. ...

4. **Purchase Order**
   4.1. The Purchaser shall follow the Supplier price lists on the supplier's website.
   4.2. The Purchaser shall present Supplier with a purchase order for the provision of Goods within 7 days of the commencement date.
5. **Service Delivery**
   5.1. ...
   5.2. The Supplier shall on receipt of a purchase order for Goods make them available within 1 days.
   5.3. If for any reason the conditions stated in 4.1 or 4.2 are not met, the Purchaser is entitled to charge the Supplier the rate of $100 for each hour the Goods are not delivered.
6. **Payments**
   6.1. The payment terms shall be in full upon receipt of invoice. Interest shall be charged at 5 % on accounts not paid within 7 days of the invoice date. Another 1.5 % interest shall be applicable if not paid within next 15 days. The prices shall be as stated in the sales order unless otherwise agreed in writing by the Supplier.
   6.2. Payments are to be sent electronically, and are to be performed under standards and guidelines outlined in PayPal.
7. **Disputes:** Omitted due to limited space.
8. **Termination:** Omitted due to limited space.

The agreement covers a range of rule objectives such as roles of the involved parties (e.g., Supplier, Purchaser), authority and jurisdiction (Australia, Queensland Courts), deontic conditions associated with roles (permissions, prohibition), and temporal properties to perform required actions. A contract can be viewed as a legal document containing a finite set of articles (where each article contains a set of clauses and subclauses). The above-discussed agreement includes two main types of clauses namely: (i) *definitional clauses*, which define the basic concepts contained in this agreement; and (ii) *normative clauses*, which regulate the actions of Purchaser and Supplier for the performance of contract, and include deontic notions e.g., obligations, permission etc.

3 Modal Defeasible Logic: An Informal Introduction

The following is a modal extension of DL, based on the work of [15,16]. The basic language is defined as follows. Given a set PROP of *propositional atoms*, the set $\mathsf{Lit} = \mathsf{PROP} \cup \{\neg p \mid p \in \mathsf{PROP}\}$ denotes the set of *literals*. If q is a literal, then $\sim q$ denotes its complement; if q is a positive literal p then $\sim q$ is $\neg p$, and if q is $\neg p$ then $\sim q$ is p. Let MOD denotes the set of modal operators. Then the set of *modal literals* is $\mathsf{ModLit} = \{\mathrm{X}l, \neg \mathrm{X}l \mid l \in \mathsf{Lit}, \mathrm{X} \in \mathsf{MOD}\}$.

We define a *defeasible theory* D as a structure $(F, R, >)$, where (i) F is a set of *facts* or indisputable statements, (ii) R is the set of rules, and (iii) $>$ is an acyclic *superiority relation* on R.

To enhance the expressiveness of a rule to encode chains of obligations and violations, following the ideas of [14], a sub-structural operator $\otimes$ is introduced to capture an obligation and the obligations arising in response to the violation of the obligation. Thus, given an expression like $a \otimes b$, the intuitive reading is that if a is possible, then a is the first choice and b is the second one; if $\neg a$ holds, i.e., a is violated, then b is the actual choice. That is, the $\otimes$-operator is used to build chains of preferences, called $\otimes$-*expression*, such that: (i) each literal is a $\otimes$-expression; (ii) if A is an $\otimes$-expression and b is a (modal) literal, then $A \otimes b$ is an $\otimes$-expression.

Hence, given Lbl a set of arbitrary labels, every rule in R is of the form $r: \ A(r) \hookrightarrow C(r)$, where:

- $r \in \mathsf{Lbl}$ is the unique identifier of the rule;
- $A(r) = \phi_1, \ldots, \phi_n$, the *antecedent* of the rule, is a finite set of (modal) literals denoting the premises of the rule;
- $\hookrightarrow \in \{\rightarrow, \Rightarrow, \rightsquigarrow\}$ denotes the *type* of the rule;
- $C(r)$ is the *consequent* (or *head*) of the rule, which can be either a single (modal) literal, or an $\otimes$-expression otherwise.

The intuition behind different arrows is the following. DL supports three types of rules namely: *strict rules* ($r : A(r) \rightarrow C(r)$), *defeasible rules* ($r : A(r) \Rightarrow C(r)$) and *defeaters* ($r : A(r) \rightsquigarrow C(r)$). Strict rules are rules in the classical sense, the conclusion follows every time the antecedents holds; a defeasible rules is allowed to assert its conclusion in case there is no contrary evidence to it. Finally, defeaters suggests there is a connection between its premises and its conclusion(s) but not strong enough to warrant the conclusion on its own; they are used to defeat rules for the opposite conclusion(s).

DL is a *skeptical* nonmonotonic logic meaning that it does not support contradictory conclusions. Instead, it seeks to resolve conflicts. In case there is some support for concluding A but there is also support for concluding $\neg A$, DL does not conclude either of them. However, if the support for A has priority over the support of $\neg A$ then A is concluded. Here, the superiority relation $>$ is used to describe the relative strength of rules on R. When $r_1 > r_2$, then r_1 is called *superior* to r_2, and r_2 *inferior* to r_1. Intuitively, $r_1 > r_2$ expresses that r_1 overrides r_2 if both rules are applicable[3].

Provability is based on the concept of *derivation* (or *proof*) in D satisfying the proof conditions. A *conclusion* of D is a tagged literal and can have one of the following forms: (i) $+\Delta q$ meaning that q is definitely provable in D (i.e., using only facts or strict rules); (ii) $-\Delta q$ meaning that q is definitely rejected in D; (iii) $+\partial q$ meaning that q is defeasibly provable in D; and (iv) $-\partial q$ meaning that q is defeasibly rejected in D.

[3] A rule is applicable if all literals in its antecedent have already been proved.

Strict derivations are obtained by forward chaining of strict rules while a defeasible conclusion p can be derived if there is a rule whose conclusion is p, and its (prerequisite) antecedent has either already been proved or given in the case at hand (i.e., facts), and any stronger rules whose conclusion is $\neg p$ has prerequisite that it failed to be derived. In other words, a conclusion p is defeasibly derivable when: (i) p is a fact; or (ii) there is an applicable strict or defeasible rule for p, and either all rules for $\neg p$ are discarded (i.e., inapplicable) or every rule for $\neg p$ is weaker than an applicable rule for p.

A full description of the proof theory can be found in [2]. A useful metaphor is to imagine, the rules with conclusion p form a team that competes with opposite team consisting of the rules with conclusion $\neg p$. If the former team wins p is defeasible provable, whereas if the opposing team wins, p is non-provable or rejected from the theory.

Throughout the paper, we use the following abbreviations on set of rules: R_s (R_d) denotes the set of strict (defeasible) rules, $R[q]$ denotes the set of rules with consequent q, and for a $r \in R$, $C(r, i)$ denotes the i^{th} (modal) literal that appears in $C(r)$.

4 LegalRuleML: The Legal Rule Markup Language

LegalRuleML [23] is a rule interchange language proposed by OASIS, which extends RuleML with features specific to legal domain [4]. It aims to bridge the gap between natural language descriptions and semantic norms [3], and can be used to model various laws, rules and regulations by translating the compliance requirements into a machine readable format [19]. Accordingly, LegalRuleML implements defeasibility as within the law where the precedent of a rule is satisfied by the facts of a case, then assumably the conclusion of the rule holds, but not necessarily [4]. The defeasibility of these legal rules can further identify exceptions and conflicts as well as mechanisms to resolve these conflicts within the norms. Additionally, LegalRuleML provides features to model various effects that follow from applying rules, such as obligations, permissions and prohibitions.

A contract written in LegalRuleML is not intended to be executed directly, but the business logic can be transformed into a target language of a rule-based system to execute. In this section we are going to explore the building blocks of LegalRuleML and propose a method to transform legal norms represented in LegalRuleML into DL theory. Since LegalRuleML is essentially an extension of RuleML, here we only highlight the differences and identify the additions to faithfully represent legal norms.

4.1 Premises and Conclusions

The first thing we have to consider is the representation of predicates (atoms) to be used in premises or conclusions in LegalRuleML. LegalRuleML extends the

construct from RuleML and represents a predicate as an n-ary relation, and is defined using an element <ruleml:Atom>[4].

Normative effects of an atom, on the other hand, are captured by embedding the atom inside a deontic element. The legal concepts such as *obligation* (<lrml:Obligation>), *permission* (<lrml:Permission>), *prohibition* (<lrml:Prohibition), and *right* (<lrml:Right>[5]) are the basic deontic elements in LegalRuleML. Further refinements are possible by: (i) providing an iri[6] attribute of a deontic specification, or (ii) using an <lrml:Association> to link a deontic specification to its meaning with the <lrml:applyModality> element.

```
<lrml:Associations>
  <lrml:Association key="asc1">
    <lrml:appliesModality iri="ex:achievementObligation"/>
    <lrml:toTarget keyref="#oblig101"/>
  </lrml:Association>
</lrml:Associations>

<lrml:Obligation key="oblig101">
  <ruleml:Atom key=":atom109">
    <ruleml:Rel iri="pay"/>
    <ruleml:Ind>Purchaser</ruleml:Ind>
    <ruleml:Ind>receivedReciept</ruleml:Ind>
    <ruleml:Ind>Supplier</ruleml:Ind>
  </ruleml:Atom>
</lrml:Obligation>
```

Accordingly, the above listing represents a modal literal OBL *pay*(*purchaser*, *receivedReceipt*, *supplier*) for the clause 6.1 in the contract that is true when *purchaser* has the (achievement) obligation[7] to pay the *supplier* upon receiving the payment[8].

4.2 Rules and Rulebases

Norms in LegalRuleML are represented as collections of statements, and can be classified into four different types according to their nature, namely: *norm statements*, *factual statements*, *override statements* and *violation-reparation statements*. These can be further classified into subtypes, as depicted in Fig. 1.

[4] Elements from LegalRuleML and elements inherited from RuleML will be prefixed with lrml and ruleml, respectively. Information about transforming norms represented using RuleML to DL can be found in [10]. The attributes key and keyref in LegalRuleML correspond to an unique identifier of a Node element and reference to a Node element, respectively.

[5] Note that the *right* here is different from the "right" in RuleML. In LegalRuleML, it is a deontic specification that gives a permission to a party and implies there is no obligation or prohibition on the other parties [23]; while "right" in RuleML means the right hand side of a rule.

[6] An iri attribute on a Node element in LegalRuleML corresponds to an <owl:sameAs> relationship in the abstract syntax.

[7] There are several types of obligations based on temporal validity and effects they produce e.g., *achievement, maintenance* etc., see [19] for details.

[8] In this paper, we are going to use the modal operator OBL for obligation, PER for permission, FOR for prohibition (forbidden).

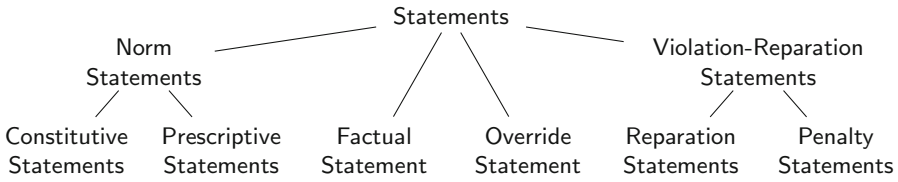

Fig. 1. Types of statements in LegalRuleML

In this section, we are going to explore different types of statements and describe how they can be transformed into rules in DL.

Norm Statements. Legal norms, in general, can be classified into *constitutive norms* (which is used to represent *institutional facts* [28] and provides definitions of terms and concepts in a jurisdiction [23]), and *prescriptive norms* (which specify the deontic behavior and effect of a legal system). These can be represented as *constitutive statements* (`<lrml:ConstitutiveStatement>`) and *prescriptive statements* in LegalRuleML (`<lrml:PrescriptiveStatement>`), respectively, to allow new information to be derived using existing rules.

The following is an example of a prescriptive statement representing the first statement of the clause 3.2 of the service contract where *goods* marked with *special order* are subject to a surcharge.

```
<lrml:PrescriptievStatement key="r1">
  <ruleml:Rule key=":ruletemplate1">
    <lrml:hasStrength>
      <lrml:DefeasibleStrength key="str1"
        iri="http://example.org/legalruleml/ontology#defeasible1"/>
    </lrml:hasStrength>
    <ruleml:if>
      <ruleml:And>
        <ruleml:Atom key=":atom2">
          <ruleml:Rel iri=":specialOrder"/>
          <ruleml:Ind>X</ruleml:Ind>
        </ruleml:Atom>
      </ruleml:And>
    </ruleml:if>
    <ruleml:then>
      <lrml:Obligation>
        <ruleml:Atom key=":atom3">
          <ruleml:Rel iri=":surcharge"/>
          <ruleml:Ind>X</ruleml:Ind>
        </ruleml:Atom>
      </lrml:Obligation>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptievStatement>
```

Similar to the *derivation rules* in RuleML, every constitutive/prescriptive statement has two parts: *conditions* (`<ruleml:if>`), which specify the conditions (using a conjunction of formulas and may possibly empty), and *conclusion* (`<ruleml:then>`), the effects of the rule. Additionally, a separate element (`<lrml:hasStrength>`) can be used to specify the strength of the rule.

Both rules can have deontic formulas as their preconditions (body). However, the difference between the two statements is in the contents of the head, where the head of a prescriptive statement is a list of deontic formulas. In contrast, the head of a constitutive statement cannot be a deontic formula [23].

In this perspective, a constitutive/prescriptive statement can be transformed into a rule of the form:

$$label: \ body \hookrightarrow head.$$

where *label* is the key of the statement, $\hookrightarrow \in \{\rightarrow, \Rightarrow, \rightsquigarrow\}$ is the rule type, *body* and *head* are the set of (modal) literals inside the `<ruleml:if>` and `<rule:then>` elements of the statement, respectively. Unless otherwise specified, due to its nature, the rule modelled using a constitutive statement will be transformed into a strict rule; while the rule modelled using prescriptive statement will be transformed into a defeasible rule. Thus, the statement above will be transformed to the defeasible rule below[9]:

$$r_1: \ specialOrder \Rightarrow \text{OBL}\, surcharge$$

Factual Statements. Factual statements in essence are the expression of facts and can be considered as a special case of norm statements without the specification of premises. They denote a simple piece of information that is deemed to be true. Below is an example of a factual statement in LegalRuleML representing the fact $premiumCustomer(JohnDoe)$, meaning that "JohnDoe" is a premium customer.

```
<lrml:FactualStatement key="fact1">
  <lrml:hasTemplate>
    <ruleml:Atom key=":atom11">
      <ruleml:Rel iri=":premiumCustomer"/>
      <ruleml:Ind iri=":JohnDoe"/>
    </ruleml:Atom>
  </lrml:hasTemplate>
</lrml:FactualStatement>
```

Override Statements. To handle defeasibility, LegalRuleML uses *override statements* (`<lrml:OverrideStatement>`) to capture the relative strength of rules that appear in the legal norms. The element `<lrml:Override>` defines the relationship of superiority such that the conclusion of $r2$ overrides the conclusion of $r1$ (where $r1$ and $r2$ are the keys of statements in the legal theory, as shown below) if both statements are applicable.

[9] Note that in some variants of DL, new types of rules can be created for the deontic operator to differentiate between normative and definitional rules [13], for instance, the rule r_1 above will becomes: $specialOrder \Rightarrow_{\text{OBL}} surcharge$. However, we do not utilize this approach here as this will limit ourselves such that only one type of modality can appear in the head of the rule. As it is possible that different logics/semantics can be used to reason on the rules generated using the constitutive and prescriptive statements, using such approach will limit the logic that we can use when reasoning the rules. For example, in our case, we can use ambiguity blocking (of DL) for the rules generated using constitutive statements and ambiguity propagation [1] for the rules generated using prescriptive statements.

Consider again clause 3.2 of the contract where a *premium customer* is exempted from the surcharge for goods marked as *'Special Orders'*, which can be modelled as the rules below.

$$r_1 : specialOrder \Rightarrow \text{OBL}\, surcharge$$
$$r_2 : specialOrder, premiumCustomer \Rightarrow \text{OBL}\, \neg surcharge$$

In the above example, the conclusion of r_2 takes the precedence over the conclusion of r_1 (as showed above) if the order was made from a *premium customer*. The following listing illustrates this using an `<lrml:OverrideStatement>` element.

```
<lrml:OverrideStatement>
  <lrml:Override over="#r2" under="#r1"/>
</lrml:OverrideStatement>
```

In DL terms, this construct defines a superiority relation between $r_2 > r_1$ where r_1 and r_2 are the rules generated using the statements $r1$ and $r2$ in the legal norms, respectively.

Violation-Reparation Statements. A *Violation-Reparation Statement* is the type of statement concerning what actions are required when an obligation is violated. LegalRuleML provides two constructs to model this, namely: *penalty statements* (`<lrml:PenaltyStatement>`) and *reparation statements* (`<lrml:ReparationStatement>`), as shown below.

```
<lrml:ReparationStatement key="reps1">
  <lrml:Reparation key="rep1">
    <lrml:appliesPenalty keyref="#pen1"/>
      <lrml:toPrescriptiveStatement
              keyref="#ps1"/>
  </lrml:Reparation>
</lrml:ReparationStatement>
```

```
<lrml:PenaltyStatement key="pen1">
  <lrml:SuborderList>
    list of deontic formulas
  </lrml:SuborderList>
</lrml:PenaltyStatement>
```

Penalty statements model sanctions and/or correction for a violation of a specified rule as outlined in the reparation statement; reparation statements bind a penalty statement to the appropriate prescriptive statement and apply the penalty when a violation occurs.

To transform these statements into DL rules, we can utilize the $\otimes$-expression that we described in Sect. 3 by appending the list of modal literals that appear in the penalty statements at the end of original rule. As an example, consider the penalty statement (in clause 6.1 of the contract) for not paying invoice within the deadline, and assume that the two model literals OBL $payWith5\,\%Interest$ and OBL $payWith6.5\,\%Interest$ are transformed from the suborder list inside the penalty statement. Then the prescriptive statement $ps1$ will be updated from

$$ps1 : goods, invoice \Rightarrow \text{OBL}\, payIn7days$$

to

$$ps1 : \; goods, invoice \Rightarrow \text{OBL}\, payIn7days \otimes \text{OBL}\, payWith5\,\%Interest \otimes \text{OBL}\, payWith6.5\,\%Interest$$

4.3 Other Constructs

Up to this point, the transformations described have been simple. However, there are other elements in LegalRuleML that are not particularly intuitive. We will highlight two of them in this section.

LegalRuleML provides two elements that can be used to determine whether an obligation or a prohibition of an object has been fulfilled (`<lrml:Compliance>`) or violated (`<lrml:Violation>`).

Definition 1 (Compliance and Violation [23]).

- *A* compliance *is an indication that an obligation has been fulfilled or a prohibition has not been violated.*
- *A* violation *is an indication that an obligation or prohibition has been violated.*

Consider the listing below which represents the rule:

$$ps2: \ \text{PER}\, rel1, \text{OBL}\, rel2 \Rightarrow \text{FOR}\, \neg rel3.$$

```
<lrml:PrescriptiveStatement key="ps2">
  <ruleml:Rule key=":ruletemplate2">
    <ruleml:if>
      <ruleml:And key=":and1">
        <lrml:Violation keyref="#ps3"/>
        <lrml:Permission>
          <ruleml:Atom key=":atom4">
            <ruleml:Rel iri=":rel1"/>
            <ruleml:Ind>X</ruleml:Ind>
          </ruleml:Atom>
        </lrml:Permission>
        <lrml:Obligation key="oblig1">
          <ruleml:Atom key=":atom5">
            <ruleml:Rel iri=":rel2"/>
            <ruleml:Ind>X</ruleml:Ind>
          </ruleml:Atom>
        </lrml:Obligation>
      </ruleml:And>
    </ruleml:if>
    <ruleml:then>
      <lrml:Prohibition key="prohib1">
        <ruleml:Neg key=":neg1">
          <ruleml:Atom key=":atom6">
            <ruleml:Rel iri=":rel3"/>
            <ruleml:Ind>X</ruleml:Ind>
          </ruleml:Atom>
        </ruleml:Neg>
      </lrml:Prohibition>
    </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

Here, we have a violation element appearing in the body as a prerequisite to activate the rule, meaning that the referenced element ($ps3$ in this case) has to be violated or the rule $ps2$ cannot not be utilised. Accordingly, we have two cases: either (i) the referenced element is a modal literal, or (ii) the referenced element is a rule.

Table 1. Requirements to determine whether a literal is compliant or violated.

	q	OBL q	FOR q
Compliance	q	OBL q, q	FOR q, $\neg$q
Violation	$\neg q$	OBL q, $\neg$q	FOR q, q

Case 1: Referenced Element Is a Literal. The former is a simple case. If the referenced element is a literal, essentially it acts as a precondition to activate the rule. It is practically the same as appending the violation (respectively, compliance) condition to the body of the rule, as shown below.

$$ps2: \text{ PER}\, rel1, \text{ OBL}\, rel2, violate(p) \Rightarrow \text{ FOR}\, \neg rel3.$$

where p is the referenced literal, $violate(p)$ (respectively $comply(p)$) is a transformation, as defined in Table 1, that transforms the (modal) literal p into a set of literals that needs to be derived in order to satisfy the condition of violation (compliance). For instance, if $ps3$ is the modal literal OBL q, then the rule $ps2$ above will be updated as follows

$$ps2: \text{ PER}\, rel1, \text{ OBL}\, rel2, \text{OBL}\, \text{q}, \neg\text{q} \Rightarrow \text{ FOR}\, \neg rel3.$$

However, the case is somewhat complex when the element appears in the head of the statement, as shown in the listing below.

```
1  <lrml:PrescriptiveStatement key="ps4">
2    <ruleml:Rule key=":ruletemplate3" keyref=":ruletemplate2">
3      <ruleml:then>
4        <lrml:SuborderList>
5          <lrml:Obligation key="obl1">
6            <ruleml:Atom key=":atom26">
7              <ruleml:Rel iri=":rel3"/>
8              <ruleml:Ind>X</ruleml:Ind>
9            </ruleml:Atom>
10         </lrml:Obligation>
11         <ruleml:And>
12           <lrml:Violation keyref="#ps5"/>
13           <lrml:Obligation key="obl2">
14             <ruleml:Atom key=":atom27">
15               <ruleml:Rel iri=":rel4"/>
16               <ruleml:Ind>X</ruleml:Ind>
17             </ruleml:Atom>
18           </lrml:Obligation>
19         </ruleml:And>
20       </lrml:SuborderList>
21     </ruleml:then>
22   </ruleml:Rule>
23 </lrml:PrescriptiveStatement>
```

Here, OBL $rel4$ (Lines 13–18) is derivable only when the modal literal OBL $rel3$ (Lines 5–10) is defeated and the reference literal $ps5$ (Line 12) is violated. However, such nested rule structure is *not* supported semantically in DL. To resolve this issue, we have to modify the statement based on its expanded form.

Definition 2 ($\otimes$-expansion). *Let $D = (F, R, >)$ be a defeasible theory, and let Σ be the language of D. We define reduct(D)=$(F, R', >')$ where for every rule*

$r \in R_d$ with a $\otimes$-expression appears in its head:

$$R' = R \setminus R_d \cup \{\ r : A(r),\ verify(c_1) \Rightarrow c_1$$
$$r' : A(r),\ violate(c_1),\ verify(c_1),\ verify(c_2) \Rightarrow c_2 \otimes \cdots \otimes c_n\}$$
$$\forall r', s' \in R', r' > s' \Leftrightarrow r, s \in R\ s.t.\ r' \in reduct(r), s' \in reduct(s), r > s.$$

where verify(p) is defined as:

$$\begin{cases} violate(e) & if\ a\ violation\ element\ is\ attached\ to\ the\ element\ p, \\ comply(e) & if\ a\ compliance\ element\ is\ attached\ to\ the\ element\ p, \\ \emptyset & otherwise. \end{cases}$$

where e is the literal referenced by the element attributed to p.

Here, we can first exclude the elements in the rule head and generate the rule based on $\otimes$-expression. Then, we can apply Definition 2 recursively to transform the generated rule into a set of rules with single literal in its head. Consequently, similar to the case discussed before, we can append the element to the body of the rule(s), where appropriate. Accordingly, the statement $ps4$ above can be transformed into the DL rules as shown below.

$$ps4_1 : A(ps4) \Rightarrow \text{OBL}\, rel3$$
$$ps4_2 : A(ps4),\ \text{OBL}\, rel3,\ \neg rel3,\ violate(ps5) \Rightarrow\ \text{OBL}\, rel4$$

Case 2: Referenced Element Is a Rule. Instead, if the referenced element is a rule, then for the case of violation, we have to verify that the rule referenced is either (i) inapplicable, i.e., there is a literal in its antecedent that is not provable; or (ii) the immediate consequent of the rule is defeated or overruled by a conflicting conclusion. While for the case of compliance, we have to verify that the referenced rule is applicable and the immediate consequent of the rule is provable[10].

Definition 3 *Let $D = (F, R, >)$ be a defeasible theory. $R^b \subseteq R$ (respectively, $R^h \subseteq R$) denotes the set of rules that contains at least one element in their body (head).*

Definition 4 (Rule Status). *Let $D = (F, R, >)$ be a defeasible theory, and let Σ be the language of D. For every $r \in R^b$, r_c denotes the rule referenced by the element. We define $verifyBody(D) = (F, R', >')$ where:*

$$R' = R \setminus R_d \cup \{\ r_c^+ : A(r_c) \Rightarrow inf(r_c),$$
$$r_c^- : \Rightarrow \neg inf(r_c),$$
$$r_{cv}^- : \neg inf(r_c) \Rightarrow violation(r_c),$$
$$r_{cc}^+ : inf(r_c),\ comply(C(r_c, 1)) \Rightarrow compliance(r_c),$$
$$r_{cv}^+ : inf(r_c),\ violate(C(r_c, 1)) \Rightarrow violation(r_c),$$
$$>' = > \cup \{r_c^+ > r_c^-\}$$

[10] In this paper, we consider only the case of weak compliance and weak violation, and verify only the first (modal) literal that appears in the head of the rule. However, the method proposed here can be extended easily to support the verification of the cases of strong compliance [19] and strong violation [12].

For each r_c, $inf(r_c)$, $\neg inf(r_c)$, $compliance(r_c)$ and $violation(r_c)$ are new atoms not in the language of the defeasible theory. $inf(r_c)$ and $\neg inf(r_c)$ are used to determine whether a rule is *in force* (applicable). If r_c is in force, we can then verify whether the first literal that appears at the head of r_c is compliant or violated (represented using the atoms $compliance(r_c)$ and $violation(r_c)$, respectively).

Similar to the case when the referenced object is a literal, depending on where the element is in the rule, we can append the compliance and violation atoms to the body and head of the rule directly. However, unlike the case where the reference element is a literal, this time we can append the atoms required directly without any transformation.

4.4 Implementation

The above transformation can be used to transform legal norms represented using LegalRuleML into DL that we can reason on. We have implemented the above transformation as an extension to the DL reasoner SPINdle [22] — an open-source, Java-based DL reasoner that supports reasoning on both standard and modal defeasible logic. Theory reasoning starts from a set of legal norms represented using LegalRuleML, i.e., a rule base, and conclusions are generated based on the semantics of DL. At the moment, various tests have been performed on some small scale LegalRuleML theories (∼10 statements per theory), and it takes, on average, 150 ms to transform a LegalRuleML theory into a SPINdle defeasible theory. Future versions will include optimization of the implementation of the transformation process so that it can handle large LegalRuleML theories in a more efficient manner.

We have also implemented the transformation in reverse direction, i.e., translate a DL theory back to LegalRuleML representation. However, as can be noticed from Sect. 3, LegalRuleML supports more features than DL, so only information about the legal norms, i.e., the rules, can appear in the translated theory.

As a remark, the transformation above is compliant with the current version of the LegalRuleML specification [3]. However, it should be noted that strange results may appear if a `<lrml:violation>` (or `<lrml:compliance>`) element appears at the head of a statement (i.e., the `<ruleml:then>` part of a statement). For instance, consider the case where a `<lrml:violation>` element appears as the only element at the head of a statement. Then, it will be transformed into a rule with no head literal, which is not correct. In the light of this, we believe that additional restriction(s) should be added to the specification in order to avoid this situation.

5 Related Works

The research in the areas of e-contracting, business process compliance and automated negotiation systems has evolved over the last few years. Several rules

modelling languages have been developed (or improved existing ones) for representing the semantics of business vocabularies, facts and business rules [9], and rules transformation techniques have emerged.

The ContractLog [25] framework for describing the formal rules based on the contract specifications for automated execution and monitoring of the service level agreements (SLAs). It combines rule-based representation of SLAs using Horn rules and Meta programming techniques alternative to contracts defined in natural language or pure programming implementations in programming languages. A rule-based technique called *SweetDeal* for representing business contracts that enables the software agent to automatically create, negotiate, evaluate and execute the contract provisions with high degree of modularity is discussed in [17]. Their technique builds upon situated courteous logic programs (SCLP) knowledge representation in RuleML, and incorporates the process knowledge descriptions whose ontologies are represented in DAML+OIL[11]. DAML+OIL representations allow handling more complex contracts with behavioural provisions that might arise during the execution of contracts. The former has to rely upon multiple formalisms to represent various types of SLA rules e.g. Horn Logic, Event-Calculus, Description Logic—whereas the latter does not consider normative effects (i.e., the approach is unable to differentiate various types of obligations such as achievement, maintenance and permissions).

Semantics of Business Vocabulary and Business Rules (SBVR) [24] is an Object Management Group (OMG) standard to represent and fomalise business ontologies, including business rules, facts and business vocabularies. It provides the basis for detailed formal and declarative specifications of business policies and includes deontic operators to represent deontic concepts e.g., obligations, permissions etc. Also, it uses the controlled natural languages to represent legal norms [9]; however, the standard has some shortcomings as the semantics for the deontic notions is underspecified. This is because SBVR is based on classical first-order-logic, which is not suitable to represent deontic notions and conflicts. Also, it cannot handle contrary-to-duty obligations as these cannot be represented by standard deontic logic (see [6] for details). The legal knowledge interchange format (LKIF), on the other hand, is an XML based interchange format language [7] that aims to provide an interchangeable format to represent legal norms in a broad range of application scenarios, especially in the context of semantic web. LKIF uses XML schemas to represent theories and arguments derived from theories, where a theory in LKIF is a set of axioms and defeasible inference rules. In addition to these, there are other XML based rule interchange format languages e.g., SWRL [20], RIF [31], WSMO [27] and OWL-S [29] (see [9] for more details on the strengths and weaknesses of these languages).

Baget et al. [5] discuss techniques for transforming existential rules into Datalog$^+$[12], RuleML and OWL 2 formats. For the transformation from Datalog$^+$ into RuleML, the authors used a fragment of Deliberation RuleML 1.01, which

[11] DAM+OIL Reference: http://www.w3.org/TR/daml+oil-reference/.

[12] Datalog$^+$: a sub-language of RuleML http://wiki.ruleml.org/index.php/Rule-Based_Data_Access#Datalog.2B.2F-.

includes positive facts, universally quantified implications, equality, falsity (and conjunctions) in the heads of implications. Whereas [30] transforms the association rules into Drool Rule Language (DRL) format using Lisp-Miner[13], and [21] proposes a model driven architecture based model to transform SBVR compliant business rules extracted from business contracts of services to compliant executable rules in formal contract language (FCL [11]). However, the former's transformation is limited only to existential rules; while the latter captures only the business rule (SBVR bears only business rules), which may or may not have legal standings. Whilst, LegalRuleML represents legal standings, the LegalRuleML's temporal notions of enforceability, efficacy and applicability cannot be represented with SBVR. In contrast, the approach proposed in this paper enables the translation of defeasible expressions, and various *deontic concepts* including the notion of *penalty* and *chain of reparations*.

6 Conclusions

In this paper, we have proposed a transformation such that (legal) norms represented using LegalRuleML can be transformed into DL which provides us a method for modeling business contracts and reasoning about them in a declarative way. Whilst LegalRuleML aims at providing specifications to legal norms that can be represented in a machine readable format, the major impedance now is the lack of dedicated and reliable infrastructure that can provide support to such capability.

As a future work, we are planning to incorporate our technique into some smart-contract enabled systems, such as Ethereum [32]. This will extend its language such that, instead of using programming logics, users can define their (smart-)contracts in a declarative manner.

References

1. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: A flexible framework for defeasible logics. In: Proceedings of the 17th National Conference on Artificial Intelligence, pp. 405–410. AAAI Press/The MIT Press (2000)
2. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Trans. Comput. Logic **2**(2), 255–286 (2001)
3. Athan, T., Boley, H., Governatori, G., Palmirani, M., Paschke, A., Wyner, A.: OASIS LegalRuleML. In: International Conference on Artificial Intelligence and Law, ICAIL 2013, Rome, Italy, pp. 3–12, 10–14 June 2013
4. Athan, T., Governatori, G., Palmirani, M., Paschke, A., Wyner, A.: LegalRuleML: design principles and foundations. In: Faber, W., Paschke, A. (eds.) Reasoning Web 2015. LNCS, vol. 9203, pp. 151–188. Springer, Heidelberg (2015)
5. Baget, J., Gutierrez, A., Leclère, M., Mugnier, M., Rocher, S., Sipieter, C.: Datalog+, RuleML and OWL 2: formats and translations for existential rules. In: RuleML 2015 Challenge, Berlin, Germany, 2–5 August 2015

[13] Lisp-Miner: http://lispminer.vse.cz.

6. Carmo, J., Jones, J.: Deontic Logic and Contrary to duties. In: Handbook of Philosophical Logic, 2nd edn., pp. 265–343. Kulwer, Dordrech (2002)
7. ESTRELLA Project: The Legal Knowledge Interchange Format (LKIF), Deliverable 4.1, European Commission (2008). http://www.estrellaproject.org/
8. Goedertier, S., Vanthienen, J.: Designing compliant business processes with obligations and permissions. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 5–14. Springer, Heidelberg (2006)
9. Gordon, T.F., Governatori, G., Rotolo, A.: Rules and norms: requirements for rule interchange languages in the legal domain. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 282–296. Springer, Heidelberg (2009)
10. Governatori, G.: Representing business contracts in RuleML. Int. J. Coop. Inf. Syst. **14**(2–3), 181–216 (2005)
11. Governatori, G., Milosevic, Z.: Dealing with contract violations: formalism and domain specific language. In: EDOC 2005, pp. 46–57. IEEE Computer Society (2005)
12. Governatori, G., Olivieri, F., Scannapieco, S., Cristani, M.: Designing for compliance: norms and goals. In: Palmirani, M. (ed.) RuleML - America 2011. LNCS, vol. 7018, pp. 282–297. Springer, Heidelberg (2011)
13. Governatori, G., Rotolo, A.: Defeasible logic: agency, intention and obligation. In: Lomuscio, A., Nute, D. (eds.) DEON 2004. LNCS (LNAI), vol. 3065, pp. 114–128. Springer, Heidelberg (2004)
14. Governatori, G., Rotolo, A.: Logic of violations: a gentzen system for reasoning with contrary-to-duty obligations. Australas. J. Logic **4**, 193–215 (2006)
15. Governatori, G., Rotolo, A.: A computational framework for institutional agency. Artif. Intell. Law **16**(1), 25–52 (2008)
16. Governatori, G., Rotolo, A.: BIO logical agents: norms, beliefs, intentions in defeasible logic. Auton. Agent. Multi-Agent Syst. **17**(1), 36–69 (2008)
17. Grosof, B., Poon, T.C.: SweetDeal: representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In: The 12th International World Wide Web Conference, pp. 340–349 (2012)
18. Grosof, B.N.: Representing e-commerce rules via situated courteous logic programs in RuleML. Electron. Commer. Res. Appl. **3**(1), 2–20 (2004)
19. Hashmi, M., Governatori, G., Wynn, M.T.: Normative requirements for regulatory compliance: an abstract formal framework. Inf. Syst. Front. **18**(3), 429–455 (2016). doi:10.1007/s10796-015-9558-1
20. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language (2004). https://www.w3.org/Submission/SWRL/
21. Kamada, A., Governatori, G., Sadiq, S.: Transformation of SBVR compliant business rules to executable FCL rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 153–161. Springer, Heidelberg (2010)
22. Lam, H.-P., Governatori, G.: The making of SPINdle. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 315–322. Springer, Heidelberg (2009)
23. OASIS LegalRuleML Technical Committee: LegalRuleML Technical Committee Specifications (2015). https://www.oasis-open.org/committees/legalruleml/charter.php, retrieved 12
24. Object Management Group (OMG): Semantics of Business Vocabulary And Rules (SBVR). OMG (2008). http://www.omg.org/spec/SBVR

25. Paschke, A., Bichler, M., Dietrich, J.B.: ContractLog: an approach to rule based monitoring and execution of service level agreements. In: Adi, A., Stoutenburg, S., Tabet, S. (eds.) RuleML 2005. LNCS, vol. 3791, pp. 209–217. Springer, Heidelberg (2005)
26. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
27. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web service modeling ontology. Appl. Ontol. **1**(1), 77–106 (2005)
28. Searle, J.R.: The Construction of Social Reality. Free Press, New York (1997)
29. The OWL Services Coalition: OWL-S 1.2 Release (2008). http://www.daml.org/services/owl-s/
30. Vojír, S., Kliegr, T., Hazucha, A., Skrabal, R., Simunek, M.: Transforming association rules to business rules: easyminer meets drools. In: Joint Proceedings of the 7th International Rule Challenge, the Special Track on Human Language Technology and the 3rd RuleML Doctoral Consortium. Seattle, USA, July 2013
31. W3C RIF Working Group: RIF: Rule Interchange Format (2005). https://www.w3.org/standards/techs/rif
32. Wood, G.: Ethereum: A Secure Decentralised Generalised Transaction Ledger (2014). http://gavwood.com/paper.pdf, Accessed December