





Bridging Between LegalRuleML and TPTP for Automated Normative Reasoning

Alexander Steen¹  and David Fuenmayor^{2,3} 

¹ University of Greifswald, Greifswald, Germany
alexander.steen@uni-greifswald.de

² University of Luxembourg, Esch-sur-Alzette, Luxembourg
david.fuenmayor@uni.lu

³ University of Bamberg, Bamberg, Germany

Abstract. LegalRuleML is a comprehensive XML-based representation framework for modeling and exchanging normative rules. The TPTP input and output formats, on the other hand, are general-purpose standards for the interaction with automated reasoning systems. In this paper we provide a bridge between the two communities by (i) defining a logic-pluralistic normative reasoning language based on the TPTP format, (ii) providing a translation scheme between relevant fragments of LegalRuleML and this language, and (iii) proposing a flexible architecture for automated normative reasoning based on this translation. We exemplarily instantiate and demonstrate the approach with three different normative logics.

Keywords: Automated reasoning · LegalRuleML · Deontic logics

1 Introduction

Automated theorem proving (ATP) systems are computer programs that, given a set A of assumptions and a conjecture C as input, try to prove that C is a logical consequence of A , i.e., that it is impossible for C to be false whenever every formula from A holds. ATP systems conduct the whole reasoning process automatically, so that no user interaction is necessary during proof search.

In normative reasoning, logical formalisms are employed to represent and reason about different notions of norms, including obligations, permissions and prohibitions. In automated normative reasoning, the goal is hence to automate the reasoning process in the context of normative discourse by employing suitable logical systems. LegalRuleML [2,3] is a comprehensive XML-based representation framework for modeling and exchanging normative rules, e.g., legal norms originating from national laws of some particular country. The LegalRuleML

The second author acknowledges financial support from the Luxembourg National Research Fund (FNR) under grant CORE AuReLeE (C20/IS/14616644).

standard comes with fine-grained and expressive means for representing (possibly legal) norms in an *isomorphic* [4] fashion with respect to their original source(s). At the same time, LegalRuleML is *semantically underspecified*, in the sense that it deliberately does not prescribe a specific logic (or semantics) in which the represented norms are to be interpreted.

LegalRuleML has been employed by Robaldo et al. to provide an exhaustive formalization of the General Data Protection Regulation (GDPR) [24]. Palmirani and Governatori combine LegalRuleML with further technologies and approaches to present an integrated framework for compliance checking with legal rules, but also focusing on GDPR applications [22]. Still, there exist only comparably few systems that, in fact, automate reasoning processes based on normative knowledge. Notable examples are provided by Liu et al. who interpret legal norms in a defeasible deontic logic and provide automation for it [19], and the SPINdle prover [17] for propositional (modal) defeasible reasoning that has been used in multiple works in the normative application domain.

In contrast, there are many general-purpose ATP systems available for classical logics, e.g., for propositional logic, first-order predicate logic, and more recently for higher-order logic. These systems are being continuously improved and are increasingly becoming more effective, as witnessed by the results of the annual ATP system competition CASC [30]. The development of general-purpose ATP systems for normative reasoning is, on the other hand, complicated by the fact that there is no single logic acting as the de-facto standard formalism for normative reasoning. In general, the design and implementation of practically effective ATP systems is a non-trivial task and very laborious; and so it is easy to see that developing custom ATP systems for each distinct normative formalism is a quite unfeasible undertaking, in particular since, as witnessed in deontic logic (see Sect. 2.2), those formalisms behave as moving targets.

In this paper, we therefore propose to employ for this task general-purpose ATP systems for classical higher-order logic, and thus to reduce normative reasoning tasks to classical ATP problems in a general way. For this, we bridge between LegalRuleML and the TPTP language standard for ATP systems [31], so that any TPTP-compliant ATP system for higher-order logic can be reused as a reasoning backend for a wide range of normative logics.

The contributions of this paper are as follows:

- We define a logic-pluralistic domain specific language (DSL) for normative reasoning with TPTP-compliant ATP systems.
- We show how the DSL can be mechanically translated into ATP reasoning problems in different concrete normative logics.
- We describe a reasoning architecture that provides flexible means of automation for these normative logics.
- We present a prototypical implementation of the whole reasoning tool chain that is available as open-source code.

The remainder of this paper is structured as follows: In Sect. 2 we briefly survey the TPTP and the LegalRuleML standards, together with a very brief exposition of deontic logics as specific systems for normative reasoning. In Sect. 3,

we discuss the role of logical pluralism in normative reasoning, which is one of the key motivations of this work. Subsequently, Sect. 4 presents the utilization of a TPTP format for normative rules. In Sect. 5 we then present a flexible and uniform approach for automating normative reasoning using general purpose ATP systems. Finally, Sect. 6 concludes and sketches further work.

An extended version of this paper is available on arXiv [27].

Related Work. Automation approaches by translation have been studied by Lam and Hashmi, where they translate LegalRuleML statements into a defeasible modal logic for which an automated reasoning tool exists [18]. Their approach is, however, fixed to one specific logic formalism as opposed to the logic-pluralistic view that we put forward in this work. Similarly, Boley et al. translate RuleML information to modal logic in TPTP format [9]. However, simple modal logics are not fully adequate for normative reasoning, see the brief discussion in Sect. 2.2. Our approach is in line with the LogiKEY methodology proposed by Benzmüller et al. [6], which makes use of expressive higher-order logics for flexibly encoding, reasoning, and experimenting with normative theories.

2 Preliminaries

2.1 The TPTP Infrastructure for ATP Systems

The *Thousands of Problems for Theorem Proving* (TPTP) library and infrastructure [31] is the core platform for contemporary ATP system development and evaluation. It provides (i) a comprehensive collection of benchmark problems for ATP systems; (ii) a set of utility tools for problem and solution inspection, pre- and post-processing, and verification; and (iii) a comprehensive syntax standard for ATP system input and output.

The TPTP specifies different ATP system languages varying in their expressivity [31]: The *first-order form* (FOF) represents unsorted first-order logic, the *typed first-order form* (TFF) represents many-sorted first-order logic, and *typed higher-order form* (THF) represents classical higher-order logic. An ATP problem generally consists of symbol declarations (if the language is typed), contextual definitions and premises of the reasoning task (usually referred to as *axioms*), and a conjecture that is to be proved or refuted in the given context. The core building block of the ATP problem files in TPTP languages are so-called *annotated formulas* of form . . .

language(*name*,*role*,*formula*[,*source*[,*annotations*]]) .

Here, *language* is a three-letter identifier for the intended language in which the annotated formula is expressed (**fof**, **tff** or **thf**). The *name* is a unique identifier for referencing to the annotated formula but has no other effect on the interpretation of it. The *role* field specifies whether the *formula* should be interpreted, among others, as an assumption (role **axiom**), a type declaration (role **type**), a definition (role **definition**) or as formula to be proved (role **conjecture**). The *formula* is an ASCII representation of the respective logical

expression, where predicate and function symbols are denoted by strings that begin with a lower-case letter, variables are denoted by strings starting with an upper-case letter, the logical connectives \neg , \wedge , \vee , \rightarrow , \leftrightarrow are represented by \sim , $\&$, $|$, \Rightarrow and \Leftrightarrow , respectively. Quantifiers \forall and \exists are expressed by $!$ and $?$, respectively, followed by a list of variables bound by it. The TPTP defines several interpreted constants starting with a $\$$ -sign, including $\$true$ and $\$false$ for truth and falsehood, respectively. In typed languages, such as TFF and THF, the type $\$i$ represents the type of individuals and $\$o$ is the type of Booleans. In TFF, explicit types of symbols may be dropped and default to n-ary function types $(\$i * \dots * \$i) > \$i$ and n-ary predicate types $(\$i * \dots * \$i) > \$o$ depending on their occurrence. Finally, the *source* and *annotations* are optional extra-logical information, e.g., about its origin, its relevance, or other properties. An example in TFF is as follows:

```
tff(union_def, axiom, ! [S, T, X]: (
    member(X, union(S,T)) <=>
    ( member(X, S) | member(X, T) ) ),
source('definitions.ax'),
[relevance(1.0)]).
```

In this example, a TFF annotated formula of name `union_def` is given that describes an axiom giving a fundamental property of set union and some auxiliary information about it. A complete description of the TPTP infrastructure and its input languages, including the syntax BNF, is provided by Sutcliffe [31] and the TPTP web page (tptp.org).

2.2 Deontic Logics and LegalRuleML

Deontic logics are logical systems intended to formally represent normative notions, such as obligations, permissions and prohibitions, their relationships, and their properties [14]. An early deontic logic, today still referred to as standard deontic logic (SDL), is based on simple modal logic **D**. In this context, the modal operators are usually denoted O (for obligation) and P (for permission), where $O\varphi \leftrightarrow \neg P\neg\varphi$ holds, and every instance of $O\varphi \rightarrow P\varphi$ is validated.

In normative reasoning contexts, usually other deontic logics are employed today. Dyadic deontic logics specifically address conditional norms of the form $O(\varphi|\psi)$ (read: *It ought to be φ given ψ*) [23], defeasible deontic logics address non-monotonic reasoning patterns with defeasible norms [15], and norm-based deontic logics model norms separately from factual expressions [20].

A prominent example illustrating the shortcomings of SDL related to conditional norms is Chisholm's paradox [13], paraphrased as follows:

Assume that your neighbors are in trouble (and you like them), then ...

- (1) *You ought to go help your neighbors.*
- (2) *If you go help your neighbors, you ought to tell them you are coming over.*
- (3) *If you do not go help your neighbors, you ought not to tell them you are coming over.*

(4) *You do not go help your neighbors.*

The sentences (1) – (4) above appear to describe a plausible situation, and, intuitively, they also constitute a both logically consistent and independent set of sentences. Hence, arguably, an adequate formalization should respect these constraints. Chisholm’s paradox here mainly serves as a running example that highlights the significant effects of interpreting normative information under different logical systems.

Table 1. Some possible formalizations of Chisholm’s paradox.

Natural language	SDL-v1	SDL-v2	SDL-v3	DDL
You ought to go help your neighbors	$O h$	$O h$	$O h$	$O h$
If you go help your neighbors, you ought to tell them you are coming over	$O(h \rightarrow t)$	$O(h \rightarrow t)$	$h \rightarrow O t$	$O(t h)$
If you do not go help your neighbors, you ought not to tell them you are coming over	$\neg h \rightarrow O \neg t$	$O(\neg h \rightarrow \neg t)$	$\neg h \rightarrow O \neg t$	$O(\neg t \neg h)$
You do not go help your neighbors	$\neg h$	$\neg h$	$\neg h$	$\neg h$

Table 1 shows several different interpretations for Chisholm’s scenario; three of them formalized using SDL, and the fourth formalized using a dyadic deontic logic (DDL) where h represents “helping your neighbors” and t represents “telling them you are coming over”. As it happens, the set of formulas corresponding to the first SDL-formalization variant (SDL-v1) is inconsistent, thus allowing the derivation of every formula and, in particular, every obligation (e.g., $O k$ where k could represent “killing your neighbor”). In fact, the next two SDL-formalizations are not logically independent, and thus inadequate, see [16, §8.5] for a discussion. In dyadic deontic logics the conditional norms from above are represented using dyadic obligation operators as in $O(t|h)$ resp. $O(\neg t|\neg h)$ instead of material implications. These logic systems are specifically conceived in order to remedy shortcomings of SDL in addressing the so-called ‘paradoxes’ of deontic logic related to conditional obligations. Unsurprisingly, no logic formalism has yet been found which successfully addresses all of the many different deontic paradoxes and deficiencies, see [16, §8] for an exhaustive overview.

LegalRuleML. [2] is a comprehensive XML-based representation framework for modeling and exchanging normative rules. It extends the general RuleML standard [10] with specialized concepts and features for normative rules, legal contexts, interpretations, etc. In LegalRuleML, conditional deontic norms are represented using specialized rules called `PrescriptiveStatements` of the form ...

```
<lrml:PrescriptiveStatement>
  <ruleml:Rule closure="universal">
    <ruleml:if> ... </ruleml:if>
```

```

    <ruleml:then> ... </ruleml:then>
  </ruleml:Rule>
</lrml:PrescriptiveStatement>

```

where both the `if`-node (the body) and the `then`-node (the head) may contain the LegalRuleML deontic operators `Obligation`, `Permission` and `Prohibition`, and combinations thereof using the usual connectives. The semantics of the deontic operators is left underspecified by LegalRuleML, so that any deontic logic may be assumed, e.g., via the `appliesModality` edge element, to interpret the represented norms. `ConstitutiveStatements` represent so-called *counts-as* norms, and they cannot have deontic operators in their head.

For a thorough introduction to LegalRuleML we refer to the literature [2,3].

2.3 Domain-Specific Languages

Domain-specific languages (DSLs) are formal languages (e.g. programming or logical languages) that have been designed for use in a particular domain. Their expressivity is deliberately restricted to allow for a higher degree of abstraction, and thus to better leverage specialized domain knowledge of their users.

DSLs can be divided into stand-alone and embedded. The former provide their own custom syntax and semantics, thus allowing for a maximal level of customization, but represent a significant implementation effort by requiring the provision of a complete compilation tool chain (parser, type-checker, etc.). The latter consist essentially in a collection of definitions encoded using a more expressive ‘host’ language; this way the existing infrastructure and tools of the host environment can be reused for the DSL. In this case we often speak of an object language (the DSL) that has been embedded into the host language.

In the context of embedded DSLs, one can further differentiate between two embedding techniques, termed *deep* and *shallow* embeddings. In a deep embedding, the terms of the object language are encoded as inductive data structures in the host language, i.e., as its abstract syntax tree (AST), and term interpretation functions (providing the semantics) can then be defined inductively, e.g. for evaluation/execution or optimization. In contrast, terms in shallow embeddings correspond to syntactic abbreviations of the host language, and thus directly encode the intended semantics of object-language expressions. Hence, evaluation in a shallow embedding corresponds to evaluation in its host language, bypassing the need for defining and inductively traversing an AST. In the context of (non-classical) logic a special technique, termed *shallow semantical embeddings* [5], has been developed to harness shallow embeddings to encode (quantified) non-classical logics into classical higher-order logic.

3 Logical Pluralism in Normative Reasoning

3.1 The Problem of Formalization

In computer science, the idea of mechanistically computing formal representations of natural language in a purely compositional way, made popular through

the seminal work of Richard Montague [21], has been pursued with the help of automated reasoners during the last thirty or so years in the area known as *computational semantics* [7]. One of the main insights has been that the expressions in natural language are semantically underspecified in the sense that not enough information can be extracted from them to construct the sort of meaning representations Montague was dreaming of, that is, formulas in some formal logical language. Thus, interpreting ordinary sentences can lead to an unfeasible number of different meaning representations [11].

Among the main determinants behind this underspecification phenomenon, we find ambiguity (syntactic and semantic) and the lack of background knowledge. Among the proposed solutions to tackle the first issue, several kinds of *underspecified semantic representations* have been proposed. However, they have been seen as challenging the application of automated reasoning methods, since disambiguation often results in different formalizations licensing disparate sets of inferences [8]. This has been seen commonly as a problem according to the traditional conception that each natural-language statement shall be correlated with one most adequate (‘correct’) formalization.

On the other hand, the available (formalized) background knowledge is also a degree of freedom determining which inferences are to be drawn from a formalized set of sentences. This knowledge can be of a linguistic nature (e.g. lexica) or more domain-specific (e.g. ontologies and knowledge bases). In fact, the availability of adequate sources for background knowledge is a well-known bottleneck in the *computational semantics* endeavor. In this respect, RuleML and related knowledge representation and interchange standards, in particular LegalRuleML, play a fundamental role in enabling the interfacing with available normative knowledge sources and ontologies.

3.2 Formalizing Normative Discourse

The problem of formalization depicted above applies notably to the logical encoding of normative discourse. This has been experienced with particular intensity in the area of deontic logic. An example of the above is the Chisholm’s paradox, as presented in Sect. 2.2, where we could appreciate how the task of adequately formalizing a set of simple natural-language sentences can give rise not only to different logical forms, but also to different ways of interpreting logical connectives, such as conditionals, (deontic) modalities, etc.

In this work, we aim at doing justice to the complex problem of formalizing normative discourse, and thus suggest to employ normative domain-specific languages (DSLs) as an intermediate representation format to encode normative knowledge in a *semantic underspecified* fashion, even reaching to the level of the logical connectives themselves, which thus require further specification for subsequent reasoning tasks. This introduces a component of *logical pluralism* into our approach, since (semantically underspecified) logical operators can (and will) be given concrete interpretations in different non-classical logics, see Sect. 5. Moreover, we aim at showing not only that such a DSL can (and should) be of

a formal logical nature, but also that it can at the same time be fully machine-readable for subsequent consumption by automated reasoning tools. Hence we introduce an illustrative normative DSL *embedded* in a suitable TPTP language below.

4 Normative Knowledge Representation in the TPTP

TPTP traditionally focused on classical logic, e.g., standards and benchmark sets for classical propositional and (first- and higher-order) predicate logic formalisms. Only recently there have been some ongoing efforts on extending TPTP towards non-classical logics as well [29]. For this purpose, the TFF language has been extended with expressions of the form ...

```
{connective_name} @ (arg1, ..., argn)
```

where *connective_name* is either a TPTP-defined name (starting with a \$ sign) or a user-defined name (starting with two \$ signs) for a non-classical operator, and the *arg_i* are terms or formulas to which the operator is applied. TPTP-defined connectives have a fixed meaning and are documented by the TPTP; the interpretation of user-defined connectives is provided by third-party systems, environments, or documentation. Non-classical operators may optionally be parameterized with key-value arguments (see below for exemplary use). The so enriched TPTP language is denoted NXF (non-classical extended first-order form). An analogous extension of THF, called NHF (non-classical higher-order form), has been introduced as well (not discussed here).

Non-classical logic languages often come with different logics (e.g., different semantics) associated with them. A prominent example are modal logic languages in which the properties of the box operator \Box depend on the concrete modal logic at hand. For example, in modal logic **S5** all instances of $\Box\varphi \rightarrow \varphi$ are tautologies, while this is not the case in modal logic **K** – still both logics share the same vocabulary. In order to resolve these ambiguities and to specify the exact logic under consideration, non-classical TPTP adds *logic specifications* to the language [29]. They are annotated formulas of form (here: in NXF) ...

```
tff(name, logic, logic_name == [options] ).
```

where *logic* is the TPTP role, *logic_name* is a TPTP-defined or user-defined designator for a logical language and *options* are comma-separated key-value pairs that fix the specific logic based on that language. Of course, changing the *logic specification* may change the provability/validity of the underlying reasoning problem.

The NXF problem representing the formalization (SDL-v3) of Chisholm's paradox, as introduced in Sect. 2.2, in simple modal logic **D** is as follows (where $\{\$box\}$ represents the modal box operator, denoted *O* in SDL):

```
tff(spec, logic, $modal == [$modalities == $modal_system_D, ...]
tff(norm1, axiom, {$box} @ (help)).
tff(norm2, axiom, help => {$box} @ (tell)).
tff(norm3, axiom, ~help => {$box} @ (~tell)).
tff(fact1, axiom, ~help).
```


The first line specifies the modal logic to be used (here, modal logic **D**), while the remaining four lines encode the formulas from Sect. 2.2. For illustration purposes, not all of the logic parameters are shown in the *logic specification*. A list of logics supported by the TPTP so far, their parameters, and their representation is available in the literature [29].

4.1 NMF: A Normative DSL in TPTP

The non-classical TPTP formats introduced above allow for encoding non-classical logics for use with generic ATP systems. Nevertheless, each problem representation in that format needs to have a fixed underlying logic as specified by the logic specification. In the present work we want to allow for working with many different normative logics in a uniform way; for this sake we introduce an *embedded DSL* (Sect. 2.3) hosted on top of non-classical TPTP formats, and referred to as *Normative Meta Form* (NMF) in the remainder. This way, every file represented in NMF will be syntactically well-formed TPTP, and hence we can use standard TPTP tools, such as syntax checkers, for processing them. Also, available software packages for ATP systems, e.g. parsers, can be reused.

More specifically, NMF extends NXF from above as follows: The operator names `$$obligation`, `$$permission`, `$$prohibition`, and `$$constitutive` are introduced. They are binary operators, and interpreted as follows ...

- `$$obligation` @ (body, head) encodes “head is obligatory given body”,
- `$$permission` @ (body, head) encodes “head is permitted given body”,
- `$$prohibition` @ (body, head) encodes “head is prohibited given body”,
- each of the three deontic operators may optionally be parameterized with the `bearer` option, e.g. `$$obligation(bearer := x) @ (body, head)`, to denote a directed deontic statement towards entity `x`, and
- `$$constitutive` @ (body, head) encodes a constitutive norm (counts-as norm) that establishes the institutional fact that `body` counts as `head`.

Since NMF extends NXF, it does not come with a fixed logic and is thus semantically underspecified. We can choose a concrete interpretation of the underspecified deontic operators using the *logic specification* as follows ...

```
tff(name, logic, $$normative == [ $$logic == target_logic ]).
```

where *target_logic* is some deontic logic identifier. We will describe the target logics currently supported in Sect. 5. For the time being, it is important to highlight that the description of the encoded norms will remain the same, regardless of which target logic we choose, and we only need to give a *logic specification* for the desired logic. Note that some deontic logics, such as SDL, do not come with built-in operators for conditional deontic expressions. Hence, our normative DSL (NMF) has been designed to abstract away the deontic operators of concrete logics, and we show in Sect. 5 how to translate from NMF to concrete deontic logics.

The running example of Chisholm’s paradox can be encoded in NMF in a logically underspecified way (i.e. without a *logic specification*) as follows:

```
tff(norm1, axiom, {$$obligation} @ ($true, help)).
tff(norm2, axiom, {$$obligation} @ (help, tell)).
tff(norm3, axiom, {$$obligation} @ (~help, ~tell)).
tff(fact1, axiom, ~help).
```

Recall that NMF is defined on top of NXF and, as such, offers first-order quantification, predicate symbols and function symbols. An even more expressive, higher-order quantified, variant of NMF could be defined analogously on top of NHF (not discussed here).

4.2 Conversion from LegalRuleML to NMF

The top-level LegalRuleML statements are translated into NMF as presented in Table 2. Note that we are currently addressing only a small fragment of LegalRuleML with this translation; many important metadata present in the LegalRuleML documentation are not yet considered. In this initial stage we primarily target automation of normative codes as formalized using deontic logics. In particular, suborder lists are currently not supported, and also strengths/exception specifications of deontic statements are not yet captured.

The translation process recursively translates the prescriptive statements, constitutive statements and factual statements of LegalRuleML into formulas in NMF. For identification purposes, key references from LegalRuleML are kept as formula names in the TPTP representation, and additional (legal) references and associations, expressed via `<lrml:LegalReferences>` or `<lrml:References>` blocks, and assigned by `<lrml:Associations>` blocks, respectively, are kept during the translation as TPTP annotations (not shown in Table 2). If a deontic operator in LegalRuleML comes with a `<lrml:Bearer>` node, this is mirrored in NMF as sketched in Sect. 4.1.

The translation from LegalRuleML to the proposed logic-pluralistic TPTP-based DSL is prototypically implemented as part of the *tptp-utils* tool, available at GitHub¹. *tptp-utils* will produce a NMF file according to the above translation scheme but without a *logic specification*. The latter can be added by the user in order to assume concrete interpretations of the normative statements, see Sect. 5. A *logic specification* could also be created automatically from the LegalRuleML document, deriving from respective `appliesModality` edges; this is an interesting venue for further work.

5 TPTP-Based Normative Reasoning Backends

The translation of LegalRuleML statements into a representation in the TPTP-based DSL introduced above does not yet allow the utilization of automated reasoning tools for automated normative reasoning. It does give, though, an abstract representation of the encoded information in a format that we can use to provide means for automation via the general TPTP automated reasoning

¹ See <https://github.com/leoprover/tptp-utils> and its README there.

Table 2. Translation scheme from a fragment of LegalRuleML to NMF. In each case except the last one the quantification closure of the formula is explicitly added to the TPTP translation; so that $\{V_1, \dots, V_n\} = \text{fv}(\text{formula}_1) \cup \text{fv}(\text{formula}_2)$ and $Q = !$ if $cl = \text{universal}$ and $Q = ?$ if $cl = \text{existential}$. The explicit quantification is omitted if $n = 0$. $tr(\cdot)$ is an adequate mapping from RuleML formulas to TPTP formulas.

LegalRuleML	NMF
<pre><lrml:PrescriptiveStatement key="id"> <ruleml:Rule closure="cl"> <ruleml:if> formula₁ </ruleml:if> <ruleml:then> <lrml:Obligation> formula₂ </lrml:Obligation> </ruleml:then> </ruleml:Rule> </lrml:PrescriptiveStatement></pre>	<pre>tff(id, axiom, Q [V₁, ..., V_n] : { \$\$obligation } @ (tr(formula₁), tr(formula₂))).</pre>
<pre><lrml:PrescriptiveStatement key="id"> <ruleml:Rule closure="cl"> <ruleml:if> formula₁ </ruleml:if> <ruleml:then> <lrml:Permission> formula₂ </lrml:Permission> </ruleml:then> </ruleml:Rule> </lrml:PrescriptiveStatement></pre>	<pre>tff(id, axiom, Q [V₁, ..., V_n] : { \$\$permission } @ (tr(formula₁), tr(formula₂))).</pre>
<pre><lrml:PrescriptiveStatement key="id"> <ruleml:Rule closure="cl"> <ruleml:if> formula₁ </ruleml:if> <ruleml:then> <lrml:Prohibition> formula₂ </lrml:Prohibition> </ruleml:then> </ruleml:Rule> </lrml:PrescriptiveStatement></pre>	<pre>tff(id, axiom, Q [V₁, ..., V_n] : { \$\$prohibition } @ (tr(formula₁), tr(formula₂))).</pre>
<pre><lrml:ConstitutiveStatement key="id"> <ruleml:Rule closure="cl"> <ruleml:if> formula₁ </ruleml:if> <ruleml:then> formula₂ </ruleml:then> </ruleml:Rule> </lrml:ConstitutiveStatement></pre>	<pre>tff(id, axiom, Q [V₁, ..., V_n] : { \$\$constitutive } @ (tr(formula₁), tr(formula₂))).</pre>
<pre><lrml:FactualStatement key="id"> formula₁ </lrml:FactualStatement></pre>	<pre>tff(id, axiom, tr(formula₁)).</pre>

infrastructure. To this end, two steps are necessary: (i) The transformation of the encoded norms into a concrete (deontic) logical formalism, and (ii) the provision of ATP systems that can reason within the respective logics.

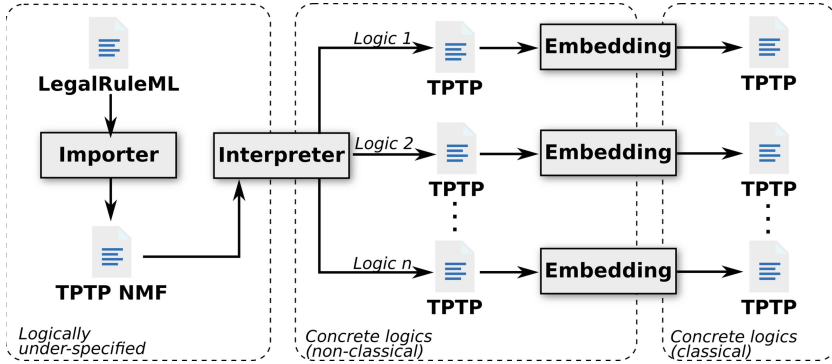


Fig. 1. Visualization of the transformation and automation process. One LegalRuleML file can be translated into multiple different TPTP reasoning problems which, in turn, can be reduced to classical reasoning problems in HOL for automation, if no special-purpose prover of the desired target logic is available.

The overall approach for logic-pluralistic automated reasoning presented in this paper is visualized in Fig. 1. The translation of LegalRuleML into NMF connects normative knowledge representation to the TPTP infrastructure, where both LegalRuleML and NMF are logically underspecified. Subsequently, the NMF representation is translated to multiple concrete logic problems formulated in (standard) non-classical TPTP. These problems are not logically underspecified anymore, as they have been encoded into specific deontic logics. Then, the resulting non-classical problems are automated using the *shallow semantical embeddings* approach [5, 25], in which the problems are encoded into classical higher-order logic (HOL). This way, general purpose HOL ATP solvers can be employed for normative reasoning. Of course, also specialized ATP systems for the respective deontic logic could be employed. However, for many quantified non-classical logics there are no ATP systems available.

The NMF representation is interpreted with respect to a concrete logic by adding a *logic specification* to it. It is of form ...

```
tff(name, logic, $$normative == [$$logic == target_logic]).
```

where *target_logic* is one of ...

- \$\$sd1, representing SDL as introduced above,
- \$\$aqvistE, representing Åqvist dyadic deontic logic **E** [1], and
- \$\$carmoJones, representing the dyadic deontic logic of Carmo and Jones [12].

Of course, this list can be extended with many more concrete logics for deontic reasoning. For the proof-of-concept presented in this paper, we restrict ourselves to these logics for the time being. An NMF problem with a *logic specification* can then be translated to a non-classical TPTP representation of the respectively chosen logic. The translation schemes for translating NMF into SDL and into DDL are presented in Tables 3 and 4.

Table 3. Translation of deontic operators from NMF to SDL based on the SDL-v3 scheme from Sect. 2.2 (narrow scope). Directed deontic operators are modeled, in each case, via $\text{\$box}(\#x)$ resp. $\text{\$dia}(\#x)$ where x is the bearer of the modality.

NMF	SDL
$\{\text{\$obligation}\} @ (\text{body}, \text{head})$	$\text{body} \Rightarrow \{\text{\$box}\} @ (\text{head})$
$\{\text{\$permission}\} @ (\text{body}, \text{head})$	$\text{body} \Rightarrow \{\text{\$dia}\} @ (\text{head})$
$\{\text{\$prohibition}\} @ (\text{body}, \text{head})$	$\text{body} \Rightarrow \{\text{\$box}\} @ (\sim\text{head})$
$\{\text{\$constitutive}\} @ (\text{body}, \text{head})$	$\text{body} \Rightarrow \text{head}$

Table 4. Translation of deontic operators from NMF to DDL. Directed deontic operators are not yet supported.

NMF	DDL
$\{\text{\$obligation}\} @ (\text{body}, \text{head})$	$\{\text{\$obl}\} @ (\text{head}, \text{body})$
$\{\text{\$permission}\} @ (\text{body}, \text{head})$	$\sim\{\text{\$obl}\} @ (\sim\text{head}, \text{body})$
$\{\text{\$prohibition}\} @ (\text{body}, \text{head})$	$\{\text{\$obl}\} @ (\sim\text{head}, \text{body})$
$\{\text{\$constitutive}\} @ (\text{body}, \text{head})$	$\text{body} \Rightarrow \text{head}$

In SDL the obligation operator is expressed using the modal logic \square operator; and the logic is specified to be modal logic **D** (as usual for SDL). Since SDL does not have any dyadic deontic operators, conditional norms are expressed via a material implication. DDL does provide a dyadic deontic operator that captures conditional norms, so the mapping is more natural here. Note that, for simplicity, the translation scheme currently follows the interpretation variant SDL-v3 (see Sect. 2.2) using a narrow-scope translation. It is planned to add further parameters to the translation so that the translation scheme can be chosen individually for each norm.

For the running example of Chisholm’s paradox, as formalized in NMF in Sect. 4.1, the concrete output for SDL as reasoning target is as follows:

```
tff(target, logic, $modal == [$quantification == $constant,
                             $constants == $rigid,
                             $modalities == $modal_system_D]).
tff(norm1-sdl, axiom, {$box} @ (help)).
tff(norm2-sdl, axiom, help => {$box} @ (tell)).
tff(norm3-sdl, axiom, ~help => {$box} @ (~tell)).
tff(fact1-sdl, axiom, ~help).
```

In Åqvist system **E** the resulting representation is (note the different order of parameters in the dyadic deontic operator) ...

```
tff(target, logic, $$ddl == [$$system == $$aqvistE]).
tff(norm1-ddl, axiom, {$$obl} @ (help,$true)).
tff(norm2-ddl, axiom, {$$obl} @ (tell,help)).
```

```
tff(norm3-ddl, axiom, {$$obl} @ (~tell,~help)).
tff(fact1-ddl, axiom, ~help).
```

For the DDL of Carmo and Jones, the output is identical except that the *logic specification* gives `$$carmoJones` instead of `$$aqvistE`. For details on the non-classical logics supported by the TPTP and the deontic logics used above, we refer to the literature [25,29]. Note that in all three cases, the problems have a fixed semantics and can thus be processed by ATP systems. The presented translation process from NMF to the deontic logics is implemented in the **LET** tool for logic embeddings [25].

In a second step, the NXF problems are embedded into classical HOL problems, represented in the THF TPTP-format. This is also done via the **LET** tool. The automation of normative reasoning via shallow embedding into HOL, as illustrated by Benzmüller et al. via their LogiKEy methodology [6], has been successful for a broad range of applications [5].

In order to provide a seamless automation process, the **LET** tool has been included into the higher-order ATP system Leo-III [26], so that the above problem statements in SDL and DDL can be given to Leo-III without the need for any external pre-processing via **LET** by the user. Unsurprisingly, Leo-III can automatically establish the unsatisfiability of the four SDL formulas `norm1-sdl`, `norm2-sdl`, `norm3-sdl` and `fact1-sdl`, thus proving their joint inconsistency; by contrast consistent conclusions can be drawn from the DDL representation.

6 Conclusion

In this paper we presented a flexible approach for using general-purpose (classical) ATP systems for normative reasoning. This is motivated, on the one hand, by the widespread availability of mature and practically effective ATP systems, and, on the other-hand, by practical challenges for providing ATP systems for the many different deontic logics employed in normative reasoning. Hence, we aim at bridging between the ATP systems community (users of the TPTP problem representation languages) and the normative knowledge representation and reasoning community (users of the LegalRuleML standard).

Our proposed approach consists in first translating a subset of LegalRuleML to a specifically crafted domain-specific language, denoted NMF, based on the TPTP standard for ATP systems. NMF is semantically underspecified and acts as an intermediate layer between natural-language representations and representations in specific logical formalisms. NMF is subsequently translated into different reasoning problems in concrete logics, represented in the recent non-classical TPTP standard. Finally, automation for non-classical TPTP is provided by *shallow semantical embeddings* into classical higher-order logic for which many different ATP systems exist. While from a purely conceptual knowledge representation perspective the intermediate NMF language might not be strictly necessary (i.e., LegalRuleML could be translated directly into concrete TPTP problem), we argue that the usage of a semantically underspecified TPTP-based

representation language comes with several pragmatic advantages with respect to practical automation. The TPTP languages are the standard formats for automated reasoning and experimentation using ATP systems, hence lowering the engineering-related barriers of providing automation for different deontic logics, and at the same time providing an abstract language for experimentation with different logics and ATP systems. In particular, the standard TPTP problem library for ATP system evaluation collects abstract problems (so-called *generators*) from which concrete reasoning tasks can be generated. The NMF layer thus connects to these efforts by providing means for the logic-pluralistic representation of domain-specific reasoning benchmarks.

The different steps in this process have been implemented as open-source tools available at GitHub. By doing so, we provided a flexible reasoning infrastructure for logic-pluralistic normative reasoning that is in line with the LogiKEY methodology [6] for designing normative theories. In contrast to LogiKEY, our focus is on flexible reasoning via ATP systems instead of enabling the interactive use of proof assistants. Two examples, one of them being the discussed Chisholm’s paradox, are available as supplemental dataset via Zenodo [28]. The dataset contains the initial LegalRuleML documents, their NMF representations and all translations to the three concrete logics.

Further Work. In this paper, we focused on three deontic logics as reasoning backends for NMF. We plan to extend the portfolio of supported deontic logics towards further relevant ones, including Input/Output logic [20]. It is planned to extend the translation tool to produce LegalRuleML output from deontic logic reasoning problems formulated in TPTP.

The DSL presented in this work is still prototypical. It does not yet capture many important aspects that are encoded in LegalRuleML documents. In particular, it is possible to extend our approach to a layered hierarchy of DSLs aiming for knowledge representation at different levels of abstraction (or domain-specificity), together with translation mechanisms for successively specifying their intended semantics. Furthermore, RuleML does allow to specify so-called *semantic profiles*. It seems a fruitful venue to closer connect these profiles to TPTP logic specifications in order to allow for a more principled approach to adjust the target logic in automation.

References

1. Åqvist, L.: Deontic logic. In: Gabbay, D., Guenther, F. (eds.) *Handbook of Philosophical Logic*, pp. 605–714. Springer, Dordrecht (1984). https://doi.org/10.1007/978-94-009-6259-0_11
2. Athan, T., Boley, H., Governatori, G., Palmirani, M., Paschke, A., Wyner, A.Z.: OASIS LegalRuleML. In: *ICAIL*, pp. 3–12. ACM (2013)
3. 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, Cham (2015). https://doi.org/10.1007/978-3-319-21768-0_6

4. Bench-Capon, T.J., Coenen, F.P.: Isomorphism and legal knowledge based systems. *Artif. Intell. Law* **1**(1), 65–86 (1992)
5. Benzmüller, C.: Universal (meta-)logical reasoning: recent successes. *Sci. Comput. Program.* **172**, 48–62 (2019)
6. Benzmüller, C., Parent, X., van der Torre, L.W.N.: Designing normative theories for ethical and legal reasoning: LogiKEy framework, methodology, and tool support. *Artif. Intell.* **287**, 103348 (2020)
7. Blackburn, P., Bos, J.: Representation and Inference for Natural Language - a First Course in Computational Semantics. CSLI Publications, Stanford (2005)
8. Blackburn, P., Kohlhase, M.: Inference and computational semantics. *J. Logic Lang. Inform.* **13**(2), 117–120 (2004)
9. Boley, H., Benzmüller, C., Luan, M., Sha, Z.: Translating higher-order modal logic from RuleML to TPTP. In: RuleML (Supplement). CEUR Workshop Proceedings, vol. 1620. CEUR-WS.org (2016)
10. Boley, H., Paschke, A., Shafiq, O.: RuleML 1.0: the overarching specification of web rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) RuleML 2010. LNCS, vol. 6403, pp. 162–178. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16289-3_15
11. Bunt, H., Muskens, R.: Computational semantics. In: Bunt, H., Muskens, R. (eds.) Computing Meaning, pp. 1–32. Springer, Dordrecht (1999). https://doi.org/10.1007/978-94-011-4231-1_1
12. Carmo, J., Jones, A.J.: Deontic logic and contrary-to-duties. In: Gabbay, D.M., Guenther, F. (eds.) Handbook of Philosophical Logic, pp. 265–343. Springer, Dordrecht (2002). https://doi.org/10.1007/978-94-010-0387-2_4
13. Chisholm, R.M.: Contrary-to-duty imperatives and deontic logic. *Analysis* **24**(2), 33–36 (1963)
14. Gabbay, D., Horty, J., Parent, X., van der Meyden, R., van der Torre, L. (eds.): Handbook of Deontic Logic and Normative Systems. College Publications, Georgia (2013)
15. Governatori, G., Rotolo, A., Calardo, E.: Possible world semantics for defeasible deontic logic. In: Ågotnes, T., Broersen, J., Elgesem, D. (eds.) DEON 2012. LNCS (LNAI), vol. 7393, pp. 46–60. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31570-1_4
16. Hilpinen, R., McNamara, P.: Deontic logic: a historical survey and introduction. In: Handbook of Deontic Logic and Normative Systems, vol. 1, pp. 3–136 (2013)
17. 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). https://doi.org/10.1007/978-3-642-04985-9_29
18. Lam, H., Hashmi, M.: Enabling reasoning with LegalRuleML. *Theory Pract. Log. Program.* **19**(1), 1–26 (2019)
19. Liu, Q., Islam, M.B., Governatori, G.: Towards an efficient rule-based framework for legal reasoning. *Knowl. Based Syst.* **224**, 107082 (2021)
20. Makinson, D., Van Der Torre, L.: Input/output logics. *J. Philos. Logic* **29**(4), 383–408 (2000)
21. Montague, R.: Universal grammar. *Theoria* **36**(3), 373–398 (1970)
22. Palmirani, M., Governatori, G.: Modelling legal knowledge for GDPR compliance checking. In: JURIX. FAIA, vol. 313, pp. 101–110. IOS Press (2018)
23. Prakken, H., Sergot, M.: Dyadic deontic logic and contrary-to-duty obligations. In: Nute, D. (ed.) Defeasible Deontic Logic, pp. 223–262. Springer, Dordrecht (1997). https://doi.org/10.1007/978-94-015-8851-5_10

24. Robaldo, L., Bartolini, C., Palmirani, M., Rossi, A., Martoni, M., Lenzini, G.: Formalizing GDPR provisions in reified I/O logic: the DAPRECO knowledge base. *J. Log. Lang. Inf.* **29**(4), 401–449 (2020)
25. Steen, A.: An extensible logic embedding tool for lightweight non-classical reasoning. In: PAAR@IJCAR. CEUR Workshop Proceedings, vol. 3201. CEUR-WS.org (2022)
26. Steen, A., Benzmüller, C.: Extensional higher-order paramodulation in Leo-III. *J. Autom. Reason.* **65**(6), 775–807 (2021). <https://doi.org/10.1007/s10817-021-09588-x>
27. Steen, A., Fuenmayor, D.: Bridging between LegalRuleML and TPTP for Automated Normative Reasoning (extended version) (2022). <https://doi.org/10.48550/arxiv.2209.05090>
28. Steen, A., Fuenmayor, D.: Supplemental data for the present article (2022). <https://doi.org/10.5281/zenodo.6702576>
29. Steen, A., Fuenmayor, D., Gleißner, T., Sutcliffe, G., Benzmüller, C.: Automated reasoning in non-classical logics in the TPTP world. In: PAAR@IJCAR. CEUR Workshop Proceedings, vol. 3201. CEUR-WS.org (2022)
30. Sutcliffe, G.: The CADE ATP system competition - CASC. *AI Mag.* **37**(2), 99–101 (2016)
31. Sutcliffe, G.: The TPTP problem library and associated infrastructure. From CNF to TH0, TPTP v6.4.0. *J. Autom. Reason.* **59**(4), 483–502 (2017)