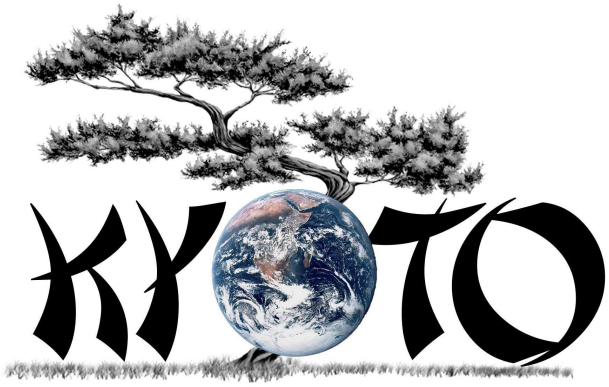
Providing First-Order Reasoning Support to Large and Complex Ontologies

Technical Report T007/WP6[a]

Version DRAFT

[a]Also submitted to the Journal of Web Semantics.

Authors: Javier Álvez[1], Paqui Lucio[1], German Rigau[1]
Affiliation: (1) EHU



Knowledge Yielding Ontologies for Transition-based Organization
ICT 211423

Grant Agreement No.	ICT 211423
Project Acronym	KYOTO
Project full title	Knowledge Yielding Ontologies for Transition-based Organization
Funding Scheme	FP7 – ICT
Project website	http://www.kyoto-project.eu/
Project Coordinator	Prof. Dr. Piek T.J.M. Vossen VU University Amsterdam Tel. + 31 (0) 20 5986466 Fax. + 31 (0) 20 5986500 Email: p.vossen@let.vu.nl
Document Number	Technical Report T007/WP6[1]
Status & version	DRAFT
Contractual date of delivery	March 18th, 2010
Actual date of delivery	April 28, 2010
Type	Report
Security (distribution level)	Public
Number of pages	43
WP contributing to the deliberable	WP6
WP responsible	Christiane Fellbaum
EC Project Officer	Werner Janusch

Authors: Javier Álvez[1], Paqui Lucio[1], German Rigau[1]

Keywords: Knowledge Representation, Ontologies, Knowledge Bases, Automatic Reasoning, Deduction, Inferencing

Abstract: In this paper, we summarize the results produced, by using first-order theorem provers as inference engines, for adapting a large and complex ontology to allow for its use in formal reasoning. In particular, our study focuses on providing first-order reasoning support to SUMO (Suggested Upper Merged Ontology). Our contribution to the area of ontological formal reasoning is threefold. Firstly, we describe our procedure for translating SUMO from its original format into the standard first order language. Secondly, we use first-order theorem provers as inference engines for debugging the ontology. Thus, we detect and repair several significant problems with the axiomatization of the SUMO ontology. Problems we encountered include incorrectly defined axioms, redundancies, non-desirable properties, and axioms that do not produce expected logical consequences. Thirdly, as a result of the process of adapting the SUMO ontology, we have discovered a basic design problem of the ontology which impedes its appropriate use with first order theorem provers. Consequently, we also propose a new transformation to overcome this limitation. As a result of this process, we obtain a validated and consistent first order version of the ontology to be used by first-order theorem provers.

Contents

1 Introduction **7**

2 Suggested Upper Merged Ontology **9**

3 First-Order Language and KIF **10**

4 First-Order Theorem Proving **12**

5 Translating Second-Order Features of SUMO **14**
5.1 First Step 14
5.2 Second Step: Row Variables 16

6 Detecting Inconsistencies **18**

7 Unfinished Definitions **22**

8 Translation of Type Information **23**
8.1 Unexpected Results from Type Information in SUMO 23
8.2 Translation of Type Information in SUMO into First-Order Formulae . . . 26
8.3 Incompatibility of Type Information with the Structure of SUMO 28

9 Adimen-SUMO: Our Proposal to Use SUMO with FOL-Theorem Provers **29**

10 Conclusions **30**

11 Acknowledgement **31**

A Analyzing Traces of FOL-Theorem Provers **35**

B Using Adimen-SUMO for Automated Reasoning **35**

C Translating SUMO into FOL-Formulae: A Detailed Example **36**

D Resources **41**

List of Tables

1 Introduction

Recently, the Semantic Web community has become very interested in having practical inference engines for foundational ontologies [7; 26; 33]. In fact, automated reasoning with ontologies is an important problem with a large practical impact. A well-known necessary condition for enabling the automated treatment of knowledge – in particular, automated reasoning with ontologies – is that ontologies must be written in a formal language whose syntax and semantics are both defined mathematically. Automated reasoning uses mechanical procedures to obtain a large body of deducible knowledge that can be inferred from a small and compact modelization. Deduction is the logical action of obtaining the statements that are true in all models of an axiomatization (or formal description).

Choosing as ontological language a known logical formalism for which automated reasoners already exist provides (all at once) the formal characterization of the language and the reasoning mechanism. Another significant feature of interest for formal ontologies is expressiveness. Since an ontology is a conceptualization of a certain domain of interest, the language should allow to express the properties that characterize that domain. It is also well-known that inference engines become increasingly complex as the underlying logic languages become more expressive. As a consequence, the trade-off between expressiveness and reasoning efficiency is a key point for the design of formal ontologies.

First-order logic (FOL) is a very well-known and quite expressive formalism and, lately, impressive progress has been made in first-order automated reasoning. In particular, there is an important collection of existing first-order theorem provers. For instance, the CASC competition[2] (see [28; 37]) evaluates the performance of sound, fully automatic, classical FOL Automated Theorem Prover systems on a bounded number of eligible problems, chosen from the TPTP Problem Library[3] (see [36]).

Ontological reasoning within the framework of first-order ontologies has been a very active area of research over the last few years. For instance, the work presented in [38] explores the feasibility of using first-order theorem provers to compute the inferences that DL (Description Logic) reasoners cannot handle. Since DL is a subset of the guarded fragment of first-order logic, suitable work on translations and interesting experiments are reported. In [29], the authors provide a translation of the Cyc ontology into FOL and report experimental results using different theorem provers (as well as the Cyc inference engine) for reasoning in the resulting first-order theory. The authors of [16] give many interesting hints for adapting the existing general purpose first-order theorem provers in order to query and check the consistency of first-order ontologies. The authors also provide two examples of inconsistencies that were detected in SUMO.[4] The main objective of [5] is to use existing logic programming model generation systems for checking the consistency of first-order ontologies. To this end, a translation from first-order ontologies into disjunctive logic programs is proposed. Finally, in [27], the authors report some preliminary experimental results evaluating the query timeout for different options when translating SUMO into

[2] `http://www.cs.miami.edu/~tptp/CASC/`

[3] `http://www.cs.miami.edu/~tptp/`

[4] `http://www.ontologyportal.org/`

FOL.[5]

Recently, this translation has been included in the CASC Competition as the SUMO Challenge. An evolved translation can be found in the TPTP Library. In the sequel, we refer to the latter translation as TPTP-SUMO.[6]

In this paper, we report on our experience using first-order theorem provers as inference engines for reasoning with large and complex ontologies. In particular, our study focuses on SUMO (Suggested Upper Merged Ontology, see [25]), although this work would have been very similar working with Cyc [22], DOLCE [13] or any other large ontologies. First, we translated SUMO from its native language into a standard first order format. SUMO is expressed in KIF (see [31]) and the syntax of KIF goes beyond the syntax of first-order logic. Therefore, our first goal has been to convert a large portion of SUMO into a first-order theory. Then, we used the first-order theorem provers as inference engines for debugging the ontology. We summarize the problems we have found and the solutions we have adopted in this process. When an inconsistency was detected, we used the explanation (or refutation) provided by the theorem prover to investigate the conflictive axioms. Once the final reason for the inconsistency was discovered, possible solutions for repairing the consistency were tested. In following this procedure, we detected several spurious items of knowledge in the ontology – that is, axioms that do not produce the expected logical consequences – as well as some non-desirable properties, incorrectly defined axioms, redundancies, etc. However, as a result of the debugging process, we also discovered a basic design problem in SUMO which impedes its appropriate use by a first order theorem prover. Thus, we also propose a further transformation to overcome this limitation. As a result of this process, we obtained a validated and consistent first order version of the ontology, hereinafter Adimen-SUMO. This version of the ontology can be used by first-order theorem provers to establish formal reasoning about the properties and relations of the classes defined by the ontology.

The outline of the paper is as follows. In Section 2, we introduce SUMO, the ontology of interest to this work. Section 3 focuses on the format used for describing SUMO. In Section 4, we describe the first-order theorem provers that we have used to debug ontologies. Section 5 explains our approach to translating second-order KIF-axioms into FOL. Once a FOL fragment of SUMO is obtained, in Section 6 we provide, as a summary, some illustrative examples of the inconsistencies discovered in SUMO by the FOL-theorem provers. However, in Section 7, we show that some parts of SUMO, in its current state, are of no use for reasoning purposes. In Section 8, we also describe that, without an additional treatment, the type information in SUMO does not produce the expected results (Subsection 8.1), and then propose a solution to this problem (Subsection 8.2). Then, in Subsection 8.3, we introduce the incompatibility of type information and the current structure of SUMO. Section 9 introduces our main result: Adimen-SUMO, which is a consistent translation of SUMO into a FOL-ontology that can be used with FOL-theorem provers for formal reasoning. Section 10 summarizes our work. Finally, for the interested

[5] In fact, as acknowledged by the authors, most of our suggestions are currently part of their translation thanks to a fruitful collaboration and discussion with them.

[6] `http://www.tptp.org`

reader, we provide four Appendixes. Appendix A gives a detailed example of how to analyze the traces obtained from FOL-theorem provers in order to debug the ontology and Appendix B presents another detailed example of the new reasoning capabilities provided by the FOL-consistent version of the ontology. Appendix C provides a thorough description of the translation of SUMO into FOL-formulae proposed in this paper. The Adimen-SUMO package[7] is described in Appendix D.

2 Suggested Upper Merged Ontology

SUMO [25] was created by the IEEE Standard Upper Ontology Working Group. Their goal was to develop a standard upper ontology to promote data interoperability, information search and retrieval, automated inference and natural language processing. SUMO provides definitions for general purpose terms and is the result of merging different free upper ontologies (e.g. Sowa's upper ontology, Allen's temporal axioms, Guarino's formal mereotopology, etc.).

SUMO consists of a set of concepts, relations, and axioms that formalize an upper ontology. An upper ontology is limited to concepts that are meta, generic, abstract or philosophical. Hence, these concepts are general enough to address (at a high level) a broad range of domain areas. Concepts that are specific to particular domains are not included in the upper ontology, but such an ontology does provide a structure upon which ontologies for specific domains (e.g. medicine, finance, engineering, etc.) can be constructed.

Currently, SUMO consists of about 20,000 terms and about 70,000 axioms when all domain ontologies are combined. However, in the work reported here, we concentrate on the upper part of the ontology. That is, on SUMO itself (file Merge.kif, version 1.61) and the mid-level ontology (file Mid-level-ontology.kif, version 1.87), which consists of about 1,000 terms and 4,000 axioms depending on the particular version.[8]

SUMO aims to provide ontological support for an increasing number of different knowledge repositories. For instance, SUMO developers also maintain a complete mapping to WordNet [25]. WordNet [12] is by far the most widely-used knowledge base. It contains manually coded information about english nouns, verbs, adjectives and adverbs, and is organized around the notion of *synset*. A synset is a set of words with the same part-of-speech that can be interchanged in a certain context. For example, ⟨*student*, *pupil*, *educatee*⟩ form a synset because they can be used to refer to the same concept. A synset is often further described by a gloss, in the case of the above synset "a learner who is enrolled in an educational institution", and by explicit semantic relations to other synsets. Each synset represents a concept which is related to other concepts by means of a large number of semantic relationships, including hypernymy/hyponymy, meronymy/holonymy, antonymy, entailment, etc. In fact, WordNet is being used world-wide to anchor different types of semantic knowledge including WordNets for languages other than English [3], domain knowledge [20] or ontologies like the EuroWordNet Top Concept Ontology [2].

[7] Available at `http://adimen.si.ehu.es/web/AdimenSUMO`.

[8] Unless explicitly stated, all the examples in this paper are extracted from those files.

Furthermore, SUMO has also been merged with YAGO [35], thus combining the rich axiomatization of SUMO with the large number of individuals acquired from Wikipedia [10]. In fact, as part of the Linking Open Data project [6], YAGO is already integrated into DBpedia [18], a large knowledge base of structured information also acquired from Wikipedia. In this way, SUMO is becoming a potentially very useful resource for improving the current automated reasoning capabilities of available intelligent web services.

As a matter of fact, the SMO category in the LTB division – first-order non-propositional theorems (axioms with a provable conjecture) from Large Theories, presented in Batches – of the CADE ATP System Competition CASC (see [28; 37]) is based on problems taken from SUMO.

3 First-Order Language and KIF

SUMO is expressed in KIF (Knowledge Interchange Format, see [31]). KIF is a language that provides for the representation of knowledge about the representation of knowledge. This allows to make all knowledge representation decisions explicit and permits the introduction of new knowledge representation constructs without changing the format. KIF has declarative semantics [14]. It is possible to understand the meaning of expressions in the language without an interpreter for manipulating those expressions. In this way, KIF differs from languages that are based on specific interpreters, such as Prolog (which are confined to Horn clauses) or Pellet (confined to OWL-DL).

Thus, KIF provides for the expression of arbitrary sentences in predicate calculus. In this sense, KIF is not exactly a first-order language. In fact, KIF can be used to write FOL-formulae, but its syntax goes beyond FOL. For instance, KIF can also be used to write second-order formulae, allowing variables representing predicates and quantification of them. Moreover, KIF allows higher-order predicates – that is, predicates having other predicates as arguments – and even formulae acting as arguments of predicates. Another non-first-order feature of KIF are the so-called *row variables*. These variables are related to infinitary logic (see [17]). Row variables allow the use of predicates and functions of arbitrary arity.

In this section, we give some examples and details of the above-mentioned features of KIF while introducing the notation of FOL-formulae that we will use in the rest of the paper. For a complete reference to KIF, the interested reader is referred to [31].

Regarding the KIF syntax, it is worth highlighting that operators are written in prefix notation and, besides, conjunction of disjunction are n-ary. The only restriction about names is that variable names always start with a question mark '?', but predicates and functions can take any name.

With respect to FOL-formulae, we use the standard notation with the following notational conventions:

- x, y and z (possibly with super-/sub-scripts) are only used for variables.
- Predicate names always start in lower case.

- There is no restriction on function names, although their notation is always inherited from KIF axioms.

Next, we illustrate the capabilities of the KIF syntax by means of some examples. First, KIF can be used to express FOL-formulae. For example, the following KIF-expression

```
(forall (?X)
  (=>
    (p ?X C)
    (not (exists (?Y) (q ?Y ?X)))))
```

corresponds to the FOL-formula

$$\forall x \ (\ p(x, C) \rightarrow \neg \exists y \ q(y, x) \ )$$

where

- C is a constant function symbol,
- p and q are predicate symbols,
- x and y are variables.

However, some KIF-axioms denote properties that cannot be expressed with FOL-formulae. For instance, the following KIF-axiom

```
(<=>
  (instance ?REL SymmetricRelation)
  (forall (?INST1 ?INST2)
    (=>
      (?REL ?INST1 ?INST2)
      (?REL ?INST2 ?INST1))))
```

does not correspond to any FOL-formula because the variable ?REL acts in the second line as an individual variable (first argument of the predicate *instance*), whereas in the last two lines it is written in the position of a predicate. Obviously, this is not allowed in FOL. Moreover, axioms are usually considered to be universally closed, which in this concrete case means that the KIF-axioms can be considered to be prefixed by **forall** (?REL). The above kind of formulae are not allowed in FOL, but are permitted in second-order logic. This second-order feature of KIF is also related to the use of the same symbol as both predicate and function, assuming an implicit connection between the two uses the same symbol. In section 5, we explain a relaxed interpretation of this second-order feature which allows to translate many KIF-axioms into FOL-formulas.

Regarding higher-order features, KIF allows to write formulas as arguments of predicates. For example, in the following axiom that expresses a temporal property

```
(<=>
  (equal (WhereFn ?THING ?TIME) ?REGION)
  (holdsDuring ?TIME (exactlyLocated ?THING ?REGION)))
```

the formula (exactlyLocated ?THING ?REGION) occurs as an argument of the predicate *holdsDuring*. This higher-order feature also allows to write axioms with the style of modal and BDI[9] logics (as introduced in [9]), such as

```
(=>
  (wants ?AGENT ?OBJ)
  (desires ?AGENT (possesses ?AGENT ?OBJ)))
```

where the formula (possesses ?AGENT ?OBJ) is an argument of the predicate *desires*. There are well-known translations of some concrete temporal logics into FOL (see, e.g., [1]) which can be taken into account, in future work, to translate this kind of KIF-axiom into FOL. However, without a suitable translation, these axioms are syntactically unacceptable to any FOL-theorem prover. Thus, we currently remove all the axioms in SUMO on which a formula appears as an argument of some predicate.

Finally, row variables in KIF serve to express properties of relations of any arbitrary arity. For example, the next KIF-axiom states that if a relation is *single-valued* then there exists a unique value that is related (as last argument) to any n-tuple:

```
(=>
  (instance ?REL SingleValuedRelation)
  (forall (@ROW ?ITEM1 ?ITEM2)
    (=>
      (and
        (?REL @ROW ?ITEM1)
        (?REL @ROW ?ITEM2))
      (equal ?ITEM1 ?ITEM2))))
```

The above (second-order) axiom does not determine the arity of ?REL, which cannot be expressed using first-order languages. This feature of KIF, when used in its whole expressiveness, relates KIF to infinitary logic (see [17]). A relation that has a non-determined arity can be taken as a variable relation and, thus, axioms with row variables may also be considered as a second-order feature of KIF. Also in Section 5, we explain how to translate axioms with row variables into FOL-formulae.

4 First-Order Theorem Proving

In this section, we briefly review the most important aspects of first-order automated reasoning and the main theorem provers that are related to our work.

[9] Belief, Desires and Intentions

Automated theorem proving is one of the most well-developed subfield of *automated reasoning*. Extensive work has been done developing techniques and tools for automatically deciding whether a set of axioms is satisfiable or if a goal follows from a set of axioms. Depending on the underlying logic, the satisfiability problem ranges from decidable (with different rates of efficiency), semi-decidable or undecidable. First-order theorem proving is one of the most mature subfields of automated theorem proving. The language of first-order logic is expressive enough to represent ontological knowledge in a reasonably natural and intuitive way. There are many fully automated systems that implement different techniques for first-order theorem proving. Most common techniques of automated theorem proving are based on refutation. Roughly speaking, this technique consists in proving that a goal ψ follows from a set of axioms Φ by proving that the conjunction $\Phi \wedge \neg\psi$ is unsatisfiable, under the premise that Φ is satisfiable. Note that any goal (also its negation) follows from an unsatisfiable (also called inconsistent) set of axioms. Hence, consistency (or satisfiability) checking is a critical task. According to this approach, consistency is proved by showing that there is no inconsistency in the set of axioms. Another technique consists in directly building a model for the given set of axioms. In this case, if there is no model, then we can decide that the set of axioms is inconsistent. The major drawback of the above two approaches comes from the fact that the satisfiability problem of FOL is semi-decidable. As a consequence, on one hand, refutation-based techniques are able to find a refutation when it does exist. Otherwise, when there is no refutation, the search space could be infinite, so the system is not able to answer. On the other hand, building a model for a satisfiable set of axioms may also be an infinite task.

There is a large library of standard benchmark examples – the *Thousands of Problems for Theorem Provers (TPTP)* (see [36]) – that has allowed significant progress on the efficiency and accuracy of the many systems implemented. The TPTP Library is used in the *CADE ATP System Competition (CASC)* (see [28; 37]), which evaluates the performance of first-order theorem provers. In this competition, some of the most successful systems are Vampire [30] and E-Prover [32] (which are refutation-based automated theorem provers), and Darwin [4], Paradox [8] and iProver [19] (which try to find a model for the given theory). Some other systems, such as MetaProver [34] and SinE[10] (the winning system of the SMO category in the 2009 edition of CASC), are based on the above systems, especially Vampire and E-Prover, but implement different resolution strategies. Apart from the CASC competition, it is worth mentioning the Mace4 and Prover9 systems, which are the successors of Mace2 [23] and Otter [24], respectively, and also MiniSat [11], which is a successful SAT solver.

In our research work, we have tested (and used) most of the above-mentioned systems, mainly the Vampire and E-Prover systems. We have translated a subset of SUMO into a set of first-order axioms which serves as input for the FOL-theorem prover. In general, we have two options when running a FOL-theorem prover regarding execution time. According to the first option, there is no time limit. Hence, if the set of axioms in the ontology is satisfiable, then the system will not terminate. According to the second option, we specify

[10] `http://www.cs.manchester.ac.uk/~hoderk/sine`

a limit on execution time and thus the system always finishes, although the answer could be "time limit expired". In both cases, if the theorem prover finds a refutation, then we have a formal proof of the existence of an inconsistency in the set of axioms which helps us debug the ontology.

5 Translating Second-Order Features of SUMO

In this section, we explain how we translate the second-order features of SUMO introduced in Section 3 into a FOL-formula.

The translation is organized in two steps. In the first step, variables acting as predicates and symbols that are used as both function and predicate are translated, as described in Subsection 5.1. In the second step, the axioms that contain row variables are translated into FOL-formulae, as explained in Subsection 5.2.

5.1 First Step

The *standard semantics* of second-order logic (SOL) interprets an n-ary predicate symbol by any possible subset of n-tuples of values of the discourse domain. Under standard semantics, SOL is incomplete. That is, there is no deductive calculus able to derive all theorems. Since FOL has a complete calculus,[11] it is obvious that SOL, under standard semantics, cannot be reducible to FOL. However, there is a well-known translation of SOL into FOL (see [21] for a detailed explanation of this translation) which preserves a non-standard semantics – called Henkin semantics – that interprets predicate symbols as any definable set of n-tuples. Therefore, SOL under Henkin semantics is less expressive than SOL under standard semantics, but it is complete. The basic idea for translating SOL-formulae into FOL-ones under Henkin semantics is to use a collection of predicates $holds_k$, where $k \geq 2$ stands for the arity of the predicate.[12] Using these predicates, any atom $P(t_1, t_2, \ldots, t_n)$ where P is a variable is translated into $holds_{n+1}(P, t_1, t_2, \ldots, t_n)$. However, in order to preserve Henkin semantics, an infinite collection of first-order axioms, called comprehension axioms, should be added to the axiomatization. Roughly speaking, comprehension axioms axiomatize the predicate $holds_k$. Of course, this is a great limitation for automated reasoning.

However, we know that ontologies are finite theories over a finite alphabet and the intended meaning of a second-order axiom like

```
(<=>
  (instance ?REL SymmetricRelation)
  (forall (?INST1 ?INST2)
    (=>
```

[11] Its semi-decidability is caused by the inexistence of an algorithm (implementing a complete deduction calculus) which always finishes stating where its input is or is not deducible.

[12] In old versions of SUMO (until Merge.kif version 1.27), a variable arity predicate *holds* was used to express second-order features.

(?REL ?INST$_1$?INST$_2$)
(?REL ?INST$_2$?INST$_1$))))

is to assert (or infer) the property

$$\forall x_1 \forall x_2 \; (\; r(x_1, x_2) \rightarrow r(x_2, x_1) \;)$$

for any predicate symbol r defined to be an instance of *SymmetricRelation* (either directly or as a logical consequence). For example, if the KIF-axiom

(instance relative SymmetricRelation)

belongs to the ontology, then we should infer that

$$\forall x_1 \forall x_2 \; (\; relative(x_1, x_2) \rightarrow relative(x_2, x_1) \;).$$

From a semantical point of view, this translation is equivalent to restricting Henkin semantical structures by considering, as a possible interpretation of predicate variables, only those relations that interpret some of the predicate symbols of the alphabet, instead of any relation definable in the domain. In some sense, this means using second-order KIF-axioms as FOL meta-axioms based on SOL syntax or axiom-schemas. For this purpose, we use the so-called *reflection axioms*, which axiomatize the predicate $holds_k$ in a finite way, by relating each predicate to a single constant symbol. In this way, we can use predicate symbols as usual, while predicates are replaced with their corresponding constant symbols when used as terms.

This is the approach that we follow in the first transformation step of our proposal, which can be described as follows (row variables are treated as standard variables).

1. We add a *reflection axiom* of the form

   $$\forall x_1 \ldots \forall x_n \; (\; r(x_1, \ldots, x_n) \leftrightarrow holds_{n+1}(r', x_1, \ldots, x_n) \;)$$

   for each n-ary predicate symbol r in the alphabet of the ontology, where r' is a new constant symbol related to r.

2. Every atom of the form

   (?REL ?INST$_1$ ... ?INST$_n$)

   in any KIF-axiom is replaced with

   (holds$_{n+1}$?REL ?INST$_1$ ... ?INST$_n$).

3. Every occurrence of a predicate symbol r acting as a term is replaced with r'.

The use of a new constant symbol r' that is related to each predicate symbol in the ontology allows us to express *reflection* using a FOL-formula while retaining its deductive power.

Applying this translation to the previous example of *SymmetricRelation*, we obtain the following FOL-formulae (note that the KIF-axioms involved are free of row variables):

$$\forall x \forall y_1 \forall y_2 \; (\; instance(x, SymmetricRelation) \leftrightarrow (holds_3(x, y_1, y_2) \rightarrow holds_3(x, y_2, y_1)) \;)$$

$$\forall x_1 \forall x_2 \; (\; holds_3(relative', x_1, x_2) \leftrightarrow relative(x_1, x_2) \;)$$

$$instance(relative', SymmetricRelation)$$

It is obvious that any first-order theorem prover can infer that

$$\forall x_1 \forall x_2 \; (\; relative(x_1, x_2) \rightarrow relative(x_2, x_1) \;)$$

from the above conjunction of formulae.

In [27], the authors also propose to use predicates $holds_k$ in order to translate SOL-axioms into FOL-formulae. However, their proposal is completely different from ours. In [27], the proposal is to use the predicates $holds_k$ to translate all the atoms. In other words, they do not restrict the use of predicates $holds_k$ to atoms with variable predicates. Thus, the only predicate symbols in the translated ontology are predicates $holds_k$. This exhaustive use of predicates $holds_k$ makes it unnecessary to provide reflection axioms, but at the same time theorem provers cannot benefit from the heuristics that are based on the defined relations.

However, in TPTP-SUMO $holds_k$ are not used since axioms with variables as predicates are used as axiom schemes. In the translation, the variable predicate is instantiated to every possible value defined in the ontology, yielding an axiom for each value. In Adimen-SUMO, we use a more compact representation that is logically equivalent to the one in TPTP-SUMO.

5.2 Second Step: Row Variables

Regarding the second transformation step, in FOL it is assumed that every predicate/function symbol is used with just one arity and, besides, predicate and function symbols are disjoint. This is just a syntactic restriction which can be satisfied when using FOL-theorem provers by just conveniently renaming predicates/functions. However, in KIF, some predicate/function symbols are implicitly used with several arities by means of row variables that can be instantiated to tuples of variables of distinct length. Let us consider the following set of axioms:

```
(<=>
  (partition @ROW)
```

$$\forall x_1 \forall x_2 \forall x_3 \; (\; partition_3(x_1, x_2, x_3) \leftrightarrow (exhaustiveDecomposition_3(x_1, x_2, x_3) \wedge disjointDecomposition_3(x_1, x_2, x_3)) \;)$$

$$\forall x_1 \forall x_2 \forall x_3 \forall x_4 \; (\; partition_4(x_1, x_2, x_3, x_4) \leftrightarrow (exhaustiveDecomposition_4(x_1, x_2, x_3, x_4) \wedge disjointDecomposition_4(x_1, x_2, x_3, x_4)) \;)$$

$$partition_3(Entity, Physical, Abstract)$$

$$partition_4(Number, RealNumber, ImaginaryNumber, ComplexNumber)$$

Figure 1: Row Variable Elimination

```
(and
  (exhaustiveDecomposition @ROW)
  (disjointDecomposition @ROW)))

(partition Entity Physical Abstract)

(partition Number RealNumber ImaginaryNumber ComplexNumber)
```

The predicate symbol *partition* is used with arities 3 and 4 in the last two axioms. Thus, the predicates in the last two axioms are distinct according to FOL. However, the definition of both predicates (with arities 3 and 4) is given by means of a single axiom (the first axiom above). This is possible in KIF thanks to the possibility of using row variables.

In TPTP-SUMO, row variables are converted into single variables, where the number of single variables varies from one to five. However, our approach to translating KIF-axioms with row variables (into FOL-formulae) uses the following iterative process, which is quite similar to one of the proposals in [27]. For each predicate symbol r used in some KIF-axiom with row variables, we proceed as follows.[13] First, we determine all the possible arities of r. The set of possible arities of a predicate symbol r is given by the arities of atoms with predicate symbol r that do not contain row variables. Then, every axiom containing atoms with predicate symbol r and one row variable is replaced with an axiom for each possible arity n of r. In the new axioms, the arity of the atom with root predicate r has to be n. For this purpose, we replace the row variable with a tuple of variables of convenient length according to n and the number of remaining terms in the atom. For example, if r is used with arity 4 and there are two more terms in the atom that contains a row variable, then the length of the tuple of variables is $4 - 2 = 2$. We proceed in this way until we eliminate all the row variables. Note that, after this transformation, it is necessary to rename the

[13] The translation of functions symbols follows the same steps.

predicate symbols that are used with several arities (if any) due to the above-mentioned FOL notational convention.

For example, the result of translating the previous set of axioms containing row variables is given in Figure 1. In these axioms, the predicate *partition* is used with arities 3 and 4 in the second and third axioms, respectively. Thus, the first axiom is replaced with two axioms, one for each arity of *partition*. Further, since @ROW is the only term in the atom (partition @ROW) of the first axiom, in the new axioms @ROW is replaced with a tuple of 3 and 4 variables, respectively.

6 Detecting Inconsistencies

A FOL-theorem prover allows to detect inconsistencies in FOL-ontologies and, furthermore, to analyze the trace of the inconsistency proof (refutation) to detect the axiom(s) that is (are) involved.

In this section, we report on our experience with detecting inconsistencies in SUMO and provide some illustrative examples.

The simplest inconsistencies we have detected are name clashes. For example, the constant *Gray* was used to denote both a color[14]

(instance Gray SecondaryColor)

and a unit of measure[15]

(instance Gray SystemeInternationalUnit).

According to SUMO knowledge, *SecondaryColor* is a subclass of *Attribute* and *SystemeInternationalUnit* is a subclass of *Quantity*. In both cases, the transitive property of subclass should be used in the inference. Besides, the classes *Attribute* and *Quantity* are inferred to be disjoint. Henceforth, the reasoner detects an inconsistency, because *Gray* cannot be a common subclass of two disjoint classes.

After submitting this inconsistency to the developers of SUMO, the inconsistency was solved in Mid-level-ontology.kif version 1.44 by replacing the above first axiom with (instance GrayColor SecondaryColor).

Another simple kind of inconsistency comes from the fact that axioms are considered to be universally closed. Consequently, a wrong name or a forgotten quantifier could cause a wrong axiom which states a property different to (and, often, stronger than) the intended one. For example, in the KIF-axiom

```
(=>
  (instance ?RESIDENCE TemporaryResidence)
  (not
    (exists (?PERSON)
      (home ?PERSON ?HOTEL))))
```

[14]Extracted from Mid-level-ontology.kif version 1.43.
[15]Extracted from Merge.kif version 1.36.

the variable ?HOTEL in the consequent should be ?RESIDENCE or vice versa.[16] The wrong version is equivalent to

$$\forall x \forall y \; (\; instance(x, TemporaryResidence) \rightarrow \neg \exists z \; home(z, y) \;)$$

whereas the correct version should be:

$$\forall x \; (\; instance(x, TemporaryResidence) \rightarrow \neg \exists z \; home(z, x) \;)$$

Obviously, the former is stronger than the latter. In fact, the inconsistency proof informed us that this wrong axiom allows to infer that

$$\forall y \forall z \; \neg home(z, y)$$

whenever some instance of *TemporaryResidence* does exist, as

(instance PalaceHotel TemporaryResidence).

A corrected version of the axiom

```
(=>
  (instance ?RESIDENCE TemporaryResidence)
  (not
    (exists (?PERSON)
      (home ?PERSON ?RESIDENCE))))
```

prevents the above inference.

A similar example can be found by analyzing the next axiom[17]

```
(=>
  (instance ?COMPANY Restaurant)
  (exists (?SERVICE ?FOOD)
    (and
      (instance ?SERVICE CommercialService)
      (agent ?SERVICE ?COMPANY)
      (instance ?SERVICE Selling)
      (instance ?BUILDING RestaurantBuilding)
      (located ?SERVICE ?BUILDING)
      (patient ?SERVICE ?FOOD)
      (instance ?FOOD Food))))
```

where the variable ?BUILDING should also be in the scope of the existential quantifier, likewise the variables ?SERVICE and ?COMPANY. However, the existential quantification of ?BUILDING has been omitted. As a consequence, ?BUILDING is considered to be universally

[16] Extracted from Merge.kif and corrected in version 1.22 as suggested to the developers of SUMO.
[17] Extracted from Mid-level-ontology.kif and corrected in version 1.19 before we detected it.

quantified and the resulting axiom is too strong in the same sense as in the previous example. In particular, note that we can infer from the above axiom that

$$\forall x \forall y \; (\; instance(x, Restaurant) \rightarrow instance(y, RestaurantBuilding) \;)$$

which allows any arbitrary instance of *RestaurantBuilding* to be asserted from any arbitrary instance of *Restaurant*. The correct axiom is obtained by replacing

exists (?SERVICE ?FOOD)

with

exists (?SERVICE ?FOOD ?BUILDING)

in the second line of the above axiom. This replacement repairs the inconsistency.

An example of another kind of more substantial or conceptual bug that can be found using theorem provers is given by the following set of axioms:[18]

1. (<=>
   (disjoint ?CLASS$_1$?CLASS$_2$)
   (**and**
   (instance ?CLASS$_1$ NonNullSet)
   (instance ?CLASS$_2$ NonNullSet)
   (**forall** (?INST)
   (**not**
   (**and**
   (instance ?INST ?CLASS$_1$)
   (instance ?INST ?CLASS$_2$))))))
2. (<=>
   (instance ?ABS Abstract)
   (**not**
   (**exists** (?POINT)
   (**or**
   (located ?ABS ?POINT)
   (time ?ABS ?POINT)))))
3. (<=>
   (instance ?PHYS Physical)
   (**exists** (?LOC ?TIME)
   (**and**
   (located ?PHYS ?LOC)
   (time ?PHYS ?TIME))))
4. (=>

[18] Axioms 1-8 are extracted from Merge.kif version 1.21 and Axioms 9-10 from Mid-level-ontology.kif version 1.26.

```
(and
  (subclass ?X ?Y)
  (instance ?Z ?X))
(instance ?Z ?Y))
```

5. (subclass NonNullSet SetOrClass)
6. (subclass Region Object)
7. (subclass Object Physical)
8. (subclass SetOrClass Abstract)
9. (disjoint Indoors Outdoors)
10. (instance Outdoors Region)

From this set of axioms, a theorem prover can deduce its falseness. Analyzing the proof provided by the system (see Appendix A), we extract the following trace:

a.	$instance(Outdoors, NonNullSet)$	[1,9]
b.	$instance(Outdoors, SetOrClass)$	[4,5,a]
c.	$instance(Outdoors, Abstract)$	[4,8,b]
d.	$\forall x\ (\ \neg located(Outdoors, x)\ )$	[2,c]
e.	$instance(Outdoors, Object)$	[4,6,10]
f.	$instance(Outdoors, Physical)$	[4,7,e]
g.	$\exists x\ (\ located(Outdoors, x)\ )$	[3,f]
h.	$\bot$	[d,g]

In summary, *Outdoors* is an instance of both *Abstract* and *Physical*, which are disjoint classes, thus yielding an inconsistency. A deeper examination shows that the problem comes from the fact that he relation *instance* is inherited through *subclass*. On one hand, *Outdoors* is an instance of *Physical* because it is also an instance of *Region*, which is a reasonable classification. On the other hand, *Outdoors* is an instance of *Abstract* because it is also an instance of *NonNullSet*. This classification is not as natural as the previous one. Further, in set theory, the empty set is always disjoint from any other set. Hence, we decided to replace Axiom 1 with the following one:[19]

```
(<=>
  (disjoint ?CLASS1 ?CLASS2)
  (forall (?INST)
    (not
      (and
        (instance ?INST ?CLASS1)
        (instance ?INST ?CLASS2)))))
```

[19] As suggested to the developers of SUMO and introduced in Merge.kif version 1.22.

This replacement repairs the inconsistency.

7 Unfinished Definitions

Usually, not all the knowledge described by an ontology is usable for automated reasoning purposes. Clear examples are axioms that use a predicate that is not entirely axiomatized in the ontology. We say that this kind of knowledge is essentially *descriptive* in order to highlight that is not usable for deduction. In addition, every predicate which uses an incompletely axiomatized predicate becomes itself incomplete.

Regarding SUMO, an example of descriptive knowledge and its propagation is given by the predicates *partition* and *disjointDecomposition*. The axiomatization of these two predicates uses the predicate *inList*, as shown in the following axioms:

```
(<=>
  (partition @ROW)
  (and
    (exhaustiveDecomposition @ROW)
    (disjointDecomposition @ROW)))

(=>
  (disjointDecomposition ?CLASS @ROW)
  (forall (?ITEM1 ?ITEM2)
    (=>
      (and
        (inList ?ITEM1 (ListFn @ROW))
        (inList ?ITEM2 (ListFn @ROW))
        (not
          (equal ?ITEM1 ?ITEM2)))
      (disjoint ?ITEM1 ?ITEM2))))
```

The axiomatization of *inList* essentially defines *inList* as an irreflexive and asymmetric binary predicate that, in terms of the function called *ListOrderFn*, is defined as follows:

```
(=>
  (inList ?ITEM ?LIST)
  (exists (?NUMBER)
    (equal (ListOrderFn ?LIST ?NUMBER) ?ITEM)))
```

In the same way, the partial binary function *ListOrderFn* is poorly axiomatized by:

```
(=>
  (and
    (instance ?LIST1 List)
    (instance ?LIST2 List)
    (forall (?NUMBER)
```

(equal (ListOrderFn ?LIST$_1$?NUMBER) (ListOrderFn ?LIST$_2$?NUMBER))))
(equal ?LIST$_1$?LIST$_2$))

This axiomatization of *ListOrderFn* is not enough for deducing its basic properties. For example, the following assertion cannot be proved:

$$\forall x_1 \forall x_2 \forall x_3 \ (\ ListOrderFn(ListFn(x_1, x_2, x_3), 2) = x_2\ )$$

As a consequence, the function *ListOrderFn* and the predicates *inList*, *partition* and *disjointDecomposition* (and several others) do not produce the expected results. In the proposal of [27] and in TPTP-SUMO, this problem remains unsolved. For example, since *partition* is only partially defined in TPTP-SUMO, the axiom

(partition Organism Animal Plant Microorganism)

does not define a real partition. In particular, an instance of *Organism* could simultaneously be an instance of both *Animal* and *Plant*. In Section 9, we explain the solution that we have implemented in Adimen-SUMO.

8 Translation of Type Information

In SUMO, there is information that describes the signature of each predicate. This type information is provided by means of the predicates *domain* and *domainSubclass* (also the predicates *range* and *rangeSubclass* for the values of functions), which associate each argument of a predicate to a class. In this way, arguments of predicates are restricted to be an instance (a subclass in the case of *domainSubclass*) of its associated class.

However, in the current state of SUMO, a direct translation of this type information does not produce the expected result, since some inconsistencies arise when using automated theorem provers. In the next subsection, we describe the problem in detail. Then, we propose an appropriate translation of type information in SUMO, which is based on the classical translation of many-sorted FOL-formulae into one-sorted FOL-formulae. In the last subsection, we describe a problem which comes from the structure of SUMO and which only arises after a proper translation of type information.

8.1 Unexpected Results from Type Information in SUMO

In this subsection, we describe the kind of inconsistencies that a FOL-theorem prover finds when type information in SUMO is directly translated. Analyzing these inconsistencies, we realize that the problem comes from a too weak axiomatization, which does not completely handle type information. The problem is illustrated using an example that has been automatically found using FOL-theorem provers.

For the explanation, we focus on the predicate *material*, which is axiomatized (including type information) in the following way:

```
(=>
  (and
    (domain ?REL ?NUMBER ?CLASS)
    (?REL @ROW))
  (instance (ListOrderFn (ListFn @ROW ?NUMBER)) ?CLASS))
(=>
  (and
    (domainSubclass ?REL ?NUMBER ?CLASS)
    (?REL @ROW))
  (subclass (ListOrderFn (ListFn @ROW ?NUMBER)) ?CLASS))
```

Figure 2: Axiomatization of *domain* and *domainSubclass* in SUMO

(a) (instance material BinaryPredicate)

(b) (domainSubclass material 1 Substance)

(c) (domain material 2 CorpuscularObject)

In other words, *material* is declared to be a binary predicate whose first argument is restricted to be a subclass of *Substance* and whose second argument has to be an instance of *CorpuscularObject*. One could expect to deduce the following statement

$$\forall x \forall y \; (\; material(x, y) \rightarrow (\; subclass(x, Substance) \wedge \\ instance(y, CorpuscularObject) \;) \;)$$

according to the intended meaning of the axiomatization of predicates *domain* and *domainSubclass* in SUMO (see Figure 2).[20]

This reasoning can be easily generalized to any axioms of the form (domain p k C) or (domainSubclass p k C). At first glance, this solution allows a proper management of type information. However, we have confirmed (using a theorem prover) that this kind of translation does not always work as one would expect. For example, as we will explain below, the translation of the following set of axioms[21] yields an inconsistency:

1. (domain temporalPart 1 TimePosition)
2. (domain temporalPart 2 TimePosition)
3. (domain time 1 Physical)
4. (domain time 2 TimePosition)
5. (<=>

[20] Recall the problems described in Section 7 about the axiomatization of *ListOrderFn*.

[21] Extracted from Merge.kif version 1.61, but we have corrected Axiom 10.

(instance ?ABS Abstract)
(**not**
(**exists** (?POINT)
(**or**
(located ?ABS ?POINT)
(time ?ABS ?POINT)))))

6. (instance temporalPart PartialOrderingRelation)
7. (<=>
(temporalPart ?POS (WhenFn ?THING))
(time ?THING ?POS))
8. (=>
(**and**
(subclass ?X ?Y)
(instance ?Z ?X))
(instance ?Z ?Y))
9. (subclass PartialOrderingRelation ReflexiveRelation)
10. (<=>
(instance ?REL ReflexiveRelation)
(**forall** (?INST)
(?REL ?INST ?INST)))

The translation of the type information in Axioms 1-4 gives the following formula:

$$\begin{aligned}&\forall x \forall y\ (\ temporalPart(x,y) \rightarrow (\ instance(x, TimePosition) \wedge \\ &\qquad\qquad\qquad\qquad instance(y, TimePosition)\ )\ ) \ \wedge \\ &\forall x \forall y\ (\ time(x,y) \rightarrow (\ instance(x, Physical) \wedge \\ &\qquad\qquad\qquad\qquad instance(y, TimePosition)\ )\ )\end{aligned}$$

In addition, the direct translation of Axiom 7 yields the formula

$$\forall x \forall y\ (\ temporalPart(x, WhenFn(y)) \leftrightarrow time(y,x)\ )$$

without using the related type information (Axioms 1-4). The problem is that the former formula, the one that comes from Axioms 1-4 and which states the type information, does not restrict the use of the latter one (from Axiom 7) for reasoning. Consequently, a theorem prover can perform the following deduction, which is wrong. First, the sentence

$$instance(temporalPart, ReflexiveRelation)$$

is a logical consequence of Axioms 6, 8, and 9. So, by Axiom 10, we get

$$\forall x\ (\ temporalPart(x,x)\ ).$$

From the above formula and Axiom 7, it follows that

$$\forall x \; (\; time(x, WhenFn(x)) \;)$$

and, by Axiom 5, we finally get

$$\forall x \; (\; \neg instance(x, Abstract) \;).$$

Then, this formula yields an inconsistency for each instance of *Abstract* defined in SUMO, e.g. $instance(YearDuration, Abstract)$.

In the next subsection, we describe a translation of type information in SUMO that solves the above problem.

8.2 Translation of Type Information in SUMO into First-Order Formulae

A suitable translation of type information in SUMO is the classical technique that transforms many-sorted FOL-formulae into equivalent one-sorted FOL ones. This technique is described in e.g. [21]. Following this proposal, we first distribute universal/existential quantification over conjunction/disjunction in every FOL-formula ϕ that results from the direct translation of KIF-axioms. Then, for each subformula $\psi = \forall x \; \alpha$ of ϕ

- if the type of x is an instance of T, then ψ is replaced with

$$\forall x \; (\; instance(x, T) \rightarrow \alpha \;).$$

- if the type of x is a

$$\forall x \; (\; subclass(x, T) \rightarrow \alpha \;).$$

Similarly, for each subformula $\psi = \exists x \; \alpha$ of ϕ

- if the type of x is an instance of T, then ψ is replaced with

$$\exists x \; (\; instance(x, T) \wedge \alpha \;).$$

- if the type of x is a subclass of T, then ψ is replaced with

$$\exists x \; (\; subclass(x, T) \wedge \alpha \;).$$

Following this transformation, the translation of Axiom 7 in Subsection 8.1 using the type information in Axioms 1-4 (also from Subsection 8.1) gives the formula

$$\forall x \forall y \; [\; (\; instance(x, TimePosition) \wedge instance(y, TimeInterval) \;) \rightarrow$$
$$(\; temporalPart(x, WhenFn(y)) \leftrightarrow time(x, y) \;) \;]$$

which is weaker than the direct translation of Axiom 7:

$$\forall x \forall y \ (\ temporalPart(x, WhenFn(y)) \leftrightarrow time(y, x)\ )$$

Indeed, thanks to the instance guards, the above formula is weak enough to do not produce any inconsistency.

For a more general example of the above-described transformation, consider the following KIF-axiom

```
(forall (?OBJ ?AGENT)
  (=>
    (exploits ?OBJ ?AGENT)
    (exists (?PROCESS)
      (and
        (agent ?PROCESS ?AGENT)
        (resource ?PROCESS ?OBJ)))))
```

and the type information associated with the three involved predicates:

1. (domain exploits 1 Object)
2. (domain exploits 2 Agent)
3. (domain agent 1 Process)
4. (domain agent 2 Agent)
5. (domain resource 1 Process)
6. (domain resource 2 Object)

By direct translation, the above KIF-axiom is converted into the FOL-formula

$$\forall x \forall y \ (\ exploits(x, y) \rightarrow \exists z \ (\ agent(z, y) \wedge resource(z, x)\ )\ ).$$

Then, by formulae 1 and 6, we know that x is restricted to be an instance of *Object*. Besides, formulae 2 and 4 restrict y to be an instance of *Agent*, and formulae 3 and 5 restrict z to be an instance of *Process*. Hence, we conveniently transform the formula into

$$\begin{aligned}\forall x \forall y \ [\ (\ instance(x, Object) \wedge instance(y, Agent)\ ) \ \rightarrow \\ (\ exploits(x, y) \ \rightarrow \ \exists z \ (\ instance(z, Process) \wedge \\ agent(z, y) \wedge resource(z, x)\ )\ )\ ].\end{aligned}$$

In [27], the problem described in Subsection 8.1 is not discussed, although the authors propose to use this technique with the aim of gaining efficiency.

Using the transformation described, we transform all the FOL-formulae that are obtained from the direct translation of KIF axioms in order to make the type information in SUMO useful. However, as we explain in the next subsection, the definition of predicates that are used in SUMO in order to structure knowledge – like *subclass*, *instance*, etc – is not suitable for dealing with type information.

8.3 Incompatibility of Type Information with the Structure of SUMO

After translating SUMO as explained in the above subsection, we realize that most of the information that is intended to be defined in SUMO cannot be inferred because of a self-reference problem. In the following example, we analyze why we cannot infer that *Object* is a subclass of *Entity* in the ontology that results from a direct translation. First, *Object* is defined as a subclass of *Physical*, and *Physical* as a subclass of *Entity*:

```
(subclass Object Physical)
(subclass Physical Entity)
```

Second, the predicate *subclass* is defined as an instance of *PartialOrderingRelation*, which is a subclass of *TransitiveRelation*:

```
(instance subclass PartialOrderingRelation)
(subclass PartialOrderingRelation TransitiveRelation)
```

The translation of the above KIF-axioms into FOL-formulae is direct. Furthermore, the next axiom establishes the relation between the *subclass* and *instance* predicates:

```
(=>
  (and
    (subclass ?X ?Y)
    (instance ?Z ?X))
  (instance ?Z ?Y))
```

Using the following type information regarding *subclass* and *instance* predicates

```
(domain subclass 1 SetOrClass)
(domain subclass 2 SetOrClass)
(domain instance 1 Entity)
(domain instance 2 SetOrClass)
```

the resulting FOL-formula is:

$$\forall x \forall y \forall z \; [\; (\; instance(x, SetOrClass) \wedge instance(y, SetOrClass) \wedge instance(z, Entity) \;) \; \rightarrow \; (\; subclass(x, y) \wedge instance(z, x) \; \rightarrow \; instance(z, y) \;) \;]$$

Since the predicate *subclass* is an instance of *PartialOrderingRelation* and, besides, *PartialOrderingRelation* is a subclass of *TransitiveRelation*, from the above formula we can infer that the predicate *subclass* is an instance of *TransitiveRelation*, provided that *subclass* is an instance of *SetOrClass*. However, the predicate *PartialOrderingRelation* is not defined in SUMO to be an instance of *SetOrClass*. Even so, there is no sense in defining the predicate

subclass in such a way.[22] Thus, we cannot infer that the predicate *subclass* is an instance of *TransitiveRelation* and, therefore, it does not follow that *Object* is a subclass of *Entity*. In this way, much other information that is supposed to be implicitly defined in SUMO cannot be inferred in practice.

A deeper analysis of this problem shows that its origin lies in the fact that SUMO is defined in terms of SUMO. That is, SUMO is self-referential in the sense that the predicates used to define SUMO are also defined using SUMO. These self-referential predicates block the deductive process. Thus, a very simple solution to this problem would be to distinguish between the meta-predicates instance and subclass, which are used for the definition of the ontology, from the predicates *instance* and *subclass*, which are defined in SUMO. Note that instance and subclass form the minimal set of predicates used for defining SUMO, since any other predicate can be defined in terms of them.

This problem is not pointed out in [27]. However, the translation in TPTP-SUMO overcomes this problem by means of an ad-hoc solution that explicitly asserts that every class defined in the ontology is an instance of *SetOrClass*.

9 Adimen-SUMO: Our Proposal to Use SUMO with FOL-Theorem Provers

As discussed in the previous section, SUMO in its current state is not suitable for use with a FOL-theorem prover since it is self-referential. Hence, in this section we propose a further transformation to overcome this limitation.

Our proposal consists in translating the whole ontology using a predefined schema. The schema includes the basic predefined predicates *instance*, *subclass*, *disjoint* and *partition*, which are defined as usual. Using this schema, we can transform the SUMO ontology into a set of FOL-axioms. In particular, we have semi-automatically translated a part of SUMO (files merge.kif version 1.61 and milo.kif version 1.87) to solve the problems described in this paper. Thus, the resulting first order version of SUMO, which we call Adimen-SUMO, is the result of applying the following treatment. First, axioms that correspond to the second-order features of SUMO have been translated as explained in Section 5. This translation includes row variables. The remaining axioms that do not correspond to FOL-formulae and that are not included in the above translation are currently discarded. Second, type information has been used as described in Section 8. The self-referential problem of SUMO introduced in Subsection 8.2 disappears by means of the predefined predicates. As a result of this process, around 88% of the original SUMO axioms have been translated.

This first order version of the SUMO ontology is almost free of inconsistencies. This version seems to contain only one inconsistency involving the following axioms (plus the definitions of *instance*, *subclass*, *disjoint* and *partition*):

[22] Naturally, *PartialOrderingRelation* is just defined as subclass of *TransitiveRelation*, *AntisymmetricRelation* and *ReflexiveRelation*.

(partition Attribute InternalAttribute RelationalAttribute)

(instance DeviceOn InternalAttribute)

(subclass NormativeAttribute RelationalAttribute)

(subclass ObjectiveNorm NormativeAttribute)

(subclass DeviceAttribute ObjectiveNorm)

(subclass DeviceStateAttribute DeviceAttribute)

(instance DeviceOn DeviceStateAttribute)

These axioms yield an inconsistency since *Attribute* is defined to be partitioned into the classes *InternalAttribute* and *RelationalAttribute*, while *DeviceOn* is defined to be an instance of both classes. Note that *DeviceOn* is a direct instance of *InternalAttribute* and is also an instance of *RelationalAttribute* by inheritance, since it is a direct instance of *DeviceStateAttribute* and the class *DeviceStateAttribute* is a subclass of *RelationalAttribute*. Our solution to this inconsistency is to remove one of the axioms that define *DeviceOn*, either (instance DeviceOn InternalAttribute) or (instance DeviceOn DeviceStateAttribute).

After correcting the previous inconsistency, no more inconsistencies arise from our current first order version of SUMO. Thus, this version of the ontology can be used by first-order theorem provers to establish formal reasoning about the properties and relations of the classes defined by the ontology. If conveniently provided to a FOL-theorem prover, our current first order version of SUMO already contains enough knowledge to infer very interesting properties. For example, it is now very straightforward to infer that plants do not have brain (or any other subclass of *AnimalAnatomicalStructure*) from the current content of the ontology. In fact, this question cannot solved using the TPTP-SUMO translation. To answer this question, it is enough to prove that

```
(not
  (and
    (instance ?BRAIN Brain)
    (instance ?PLANT Plant)
    (part ?BRAIN ?PLANT)))
```

follows from Adimen-SUMO. In Appendix B, we provide a full demonstration obtained with a FOL-theorem prover when proving the above statement.

10 Conclusions

Ontologies provide computer-accessible descriptions of the meaning of relevant concepts and relationships between these concepts. Semantic Web development requires the ability to infer implied information from formal ontologies. That is, beyond the literal meaning expressed in the ontologies, an intelligent web service system needs to know what the implications of that meaning are.

Knowledge representation is a very old field in Artificial Intelligence. The main problem is that there is a tight connection between the computational model for representing knowledge and the inferencing capabilities supported by the model. Today, the family of Web Ontology Languages, including OWL-DL [15], is the most common formal knowledge representation model, being accepted and standardized by the W3C (World Wide Web Consortium). OWL-DL is a very powerful knowledge representation model, which allows a trade-off of expressiveness versus computational plausibility and complexity. However, state-of-the-art machinery like formal reasoners such as Pellet or Fact++ are unable to cope with complex ontologies such as SUMO. Thus, as stated in [16], the application of first-order automated theorem provers to ontologies like SUMO is of crucial importance to the development of the Semantic Web infrastructure.

In this paper, we have summarized our experience with using first-order theorem provers as inference engines for debugging large and complex ontologies. In particular, we have concentrated our efforts on studying, revising and improving SUMO.

First, we have explained our approach to translating KIF-axioms into the first-order logic language. Then, we have used first-order theorem provers as inference engines for debugging the ontology. We have summarized the problems we have found and the solutions we have adopted in this process. As a result, we have obtained a validated and consistent first-order version of SUMO,[23] which can be used appropriately by first-theorem provers for automatic reasoning purposes. Following the same approach presented in this paper, we plan to continue revising the latest SUMO versions and also their domain extensions.

Although, our study focuses on SUMO (Suggested Upper Merged Ontology, see [25]), the work we reported here would be very similar working with any other large and complex ontology (i.e. Cyc or DOLCE). Thus, we are continuing this research on automated reasoning with large and complex ontologies like SUMO, DOLCE and Cyc. Our final goal is to provide formal underpinnings and efficient automated reasoning services for the Semantic Web by using expressive ontologies.

11 Acknowledgement

Partial support provided by FORMALISM TIN2007-66523, KNOW2 TIN2009-14715-C04-04, LOREA GIU07/35 and KYOTO ICT-2007-211423.

References

[1] M. Abadi, The power of temporal proofs, Theor. Comput. Sci. 65 (1) (1989) 35–83.

[2] J. Álvez, J. Atserias, J. Carrera, S. Climent, E. Laparra, A. Oliver, G. Rigau, Complete and consistent annotation of WordNet using the Top Concept Ontology, in: N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odjik, S. Piperidis, D. Tapias (eds.), Pro-

[23] Available at `http://adimen.si.ehu.es/web/AdimenSUMO`.

ceedings of the Sixth International Language Resources and Evaluation (LREC'08), European Language Resources Association (ELRA), Marrakech, Morocco, 2008.

[3] J. Atserias, G. Rigau, L. Villarejo, Spanish WordNet 1.6: Porting the spanish WordNet across Princeton versions, in: Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04), 2004.

[4] P. Baumgartner, A. Fuchs, C. Tinelli, Implementing the Model Evolution Calculus, in: S. Schulz, G. Sutcliffe, T. Tammet (eds.), Special Issue of the International Journal of Artificial Intelligence Tools (IJAIT), vol. 15(1) of International Journal of Artificial Intelligence Tools, 2005, preprint.

[5] P. Baumgartner, F. M. Suchanek, Automated reasoning support for first-order ontologies, in: J. Alferes, J. Bailey, W. May, U. Schwertel (eds.), Proceedings of the 4th International Workshop in Principles and Practice of Semantic Web Reasoning (PPSWR 2006), Revised Selected Papers, vol. 4187 of LNAI, Springer, 2006.

[6] C. Bizer, T. Heath, K. Idehen, T. Berners-Lee, Linked data on the web (ldow2008), in: Proceeding of the 17th international conference on World Wide Web (WWW '08), ACM, New York, NY, USA, 2008.

[7] B. Chandrasekaran, J. R. Josephson, V. R. Benjamins, What are ontologies, and why do we need them?, IEEE Intelligent Systems 14 (1) (1999) 20–26.

[8] K. Claessen, N. Sörensson, New techniques that improve MACE-style model finding, in: Proc. of Workshop on Model Computation (MODEL), 2003.

[9] P. R. Cohen, H. J. Levesque, Persistence, intention, and commitment, in: P. R. Cohen, J. Morgan, M. E. Pollack (eds.), Intentions in Communication, MIT Press, Cambridge, MA, 1990, pp. 33–69.

[10] G. de Melo, F. Suchanek, A. Pease, Integrating YAGO into the Suggested Upper Merged Ontology, in: Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2008), vol. 1, IEEE Computer Society, Dayton, OH, USA, 2008.

[11] N. Eén, N. Sörensson, An extensible sat-solver, Theory and Applications of Satisfiability Testing (2004) 502–518.

[12] C. Fellbaum (ed.), WordNet. An Electronic Lexical Database, The MIT Press, 1998.

[13] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, L. Schneider, Sweetening ontologies with DOLCE, in: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW '02), Springer-Verlag, London, UK, 2002.

[14] P. Hayes, C. Menzel, A semantics for the knowledge interchange format, in: In IJCAI 2001 Workshop on the IEEE Standard Upper Ontology, 2001.

[15] I. Horrocks, P. Patel-Schneider, Reducing OWL entailment to description logic satisfiability, J. of Web Semantics 1 (4) (2004) 345–357.

[16] I. Horrocks, A. Voronkov, Reasoning support for expressive ontology languages using a theorem prover, in: J. Dix, S. J. Hegner (eds.), Proceedings of the 4th International Symposium in Foundations of Information and Knowledge Systems (FoIKS 2006), vol. 3861 of Lecture Notes in Computer Science, 2006.

[17] H. J. Keisler, Model theory for infinitary logic; logic with countable conjunctions and finite quantifiers., North-Holland Pub. Co., Amsterdam, 1971.

[18] G. Kobilarov, C. Bizer, S. Auer, J. Lehmann, Dbpedia, a linked data hub and data source for web applications and enterprises, in: 18th International World Wide Web Conference, 2009.

[19] K. Korovin, iProver – an instantiation-based theorem prover for first-order logic (system description), in: A. Armando, P. Baumgartner, G. Dowek (eds.), Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR 2008), vol. 5195 of Lecture Notes in Computer Science, Springer, 2008.

[20] B. Magnini, G. Cavagliá, Integrating subject field codes into wordnet, in: Proceedings of the Second International Conference on Language Resources and Evaluation (LREC 2000), 2000.

[21] M. Manzano, Extensions of first order logic, Cambridge University Press, New York, NY, USA, 1996.

[22] C. Matuszek, J. Cabral, M. Witbrock, J. DeOliveira, An introduction to the syntax and content of cyc, in: Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, 2006.

[23] W. McCune, Mace 2.0 reference manual and guide, Tech. Rep. ANL/MCS-TM-249, Mathematics and Computer Science Division, Argonne National Laboratory (June 2001).

[24] W. McCune, L. Wos, Otter - the CADE-13 competition incarnations, J. Autom. Reasoning 18 (2) (1997) 211–220.

[25] I. Niles, A. Pease, Towards a standard upper ontology, in: C. Welty, B. Smith (eds.), Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), 2001.

[26] N. F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, M. A. Musen, Creating semantic web contents with Protégé-2000, IEEE Intelligent Systems 16 (2) (2001) 60–71.

[27] A. Pease, G. Sutcliffe, First order reasoning on a large ontology., in: G. Sutcliffe, J. Urban, S. Schulz (eds.), ESARLT, vol. 257, 2007.

[28] F. Pelletier, G. Sutcliffe, C. Suttner, The development of CASC, AI Communications 15 (2-3) (2002) 79–90.

[29] D. Ramach, R. P. Reagan, K. Goolsbey, First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology, in: In Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications, 2005.

[30] A. Riazanov, A. Voronkov, The design and implementation of VAMPIRE, AI Communications 15 (2-3) (2002) 91–110.

[31] M. G. Richard, R. E. Fikes, R. Brachman, T. Gruber, P. Hayes, R. Letsinger, V. Lifschitz, R. Macgregor, J. Mccarthy, P. Norvig, R. Patil, Knowledge interchange format version 3.0 reference manual (1992).

[32] S. Schulz, E – a brainiac theorem prover, Journal of AI Communications 15 (2/3) (2002) 111–126.

[33] S. Staab, R. Studer (eds.), Handbook on ontologies (2nd edition), International Handbooks in Information Systems, Springer, 2009.

[34] M. J. Streeter, D. Golovin, S. F. Smith, Combining multiple heuristics online, in: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada, 2007.

[35] F. M. Suchanek, G. Kasneci, G. Weikum, Yago: A Core of Semantic Knowledge, in: 16th international World Wide Web conference (WWW 2007), ACM Press, New York, NY, USA, 2007.

[36] G. Sutcliffe, C. Suttner, The TPTP Problem Library: CNF Release v1.2.1, Journal of Automated Reasoning 21 (2) (1998) 177–203.

[37] G. Sutcliffe, C. Suttner, The State of CASC, AI Communications 19 (1) (2006) 35–48.

[38] D. Tsarkov, A. Riazanov, S. Bechhofer, I. Horrocks, Using Vampire to reason with OWL, in: International Semantic Web Conference, 2004.

A Analyzing Traces of FOL-Theorem Provers

In this section, we show the trace obtained from E-Prover with an example from Section 6. More specifically, we use the set of axioms that helped us to discover the incorrect axiomatization of *disjoint*, which was corrected in Merge.kif version 1.22. The trace can be reproduced with the file **disjoint.eprover.tstp** in the Adimen-SUMO package using the following command:

```
eprover -xAuto -tAuto --tstp-in -l 6
     disjoint.eprover.tstp | epclextract
```

In Figure 3, we summarize the trace that is obtained using the above command. The whole trace can be consulted in the file **output.disjoint.eprover.txt** in the Adimen-SUMO package. The ten first steps are the axioms that will be used in the trace. These axioms are simplified using some general transformations, such as Skolemization or distribution. Besides, axioms are transformed into conjunctive normal form (or CNF), where each disjunct is represented as a list of atoms (separated by commas and delimited by square brackets), positive atoms are marked with '`++`' and negative (or negated) atoms with '`--`'. After this initial transformation, the formulae in steps 18, 28, 35 and 39 are disjuncts obtained from the transformation of the formulae in steps 1-4, where the disjuncts will not be used in the trace has been discarded by E-Prover. Note that `esk3_1` (in step 35) is a unary Skolem function that comes from the variable `X7` of the formula in step 3. Besides, the formulae in steps 40-46 are directly obtained from steps 5-10.

Then, E-Prover starts to obtain new formulae by resolution. On one hand, the formula `++instance(outdoors,nonNullSet)` (in step 71) follows from `++instance(X2,nonNullSet)` (in step 18) and `++disjoint(indoors,outdoors)` (in step 44). On the other hand, `--instance(outdoors,nonNullSet)` (in step 179) is obtained after 10 resolution steps. To sum up, from the formulae in steps 39, 41 and 42 it follows that everything is either `physical` or not `region` (in step 154) after two resolution steps. Hence, since `++instance(outdoors,region)` (from step 45), we have that `++instance(outdoors,physical)` (in step 164). Besides, from the formulae in steps 28 and 35, we have that everything is either not `abstract` or not `physical` (in step 78). Therefore, it follows that `--instance(outdoors,abstract` (in step 172). Similarly, from the formulae in steps 39, 40 and 43, it follows that everything is either `abstract` or not `nonNullSet` (in step 130) after two resolution steps. Finally, from the last two results, it follows that `--instance(outdoors,nonNullSet)` (in step 179). Hence, the formulae in steps 71 and 179 yield an inconsistency.

B Using Adimen-SUMO for Automated Reasoning

In this section, we show the trace that is obtained from E-Prover with the example in Section 9. In Figure 4 is found a brief summary of the proof that is obtained using the file **brain.eprover.tstp** from the Adimen-SUMO package. The whole proof is available

in file **output.brain.eprover.txt**. For brevity, we have used some name abbreviations: the predicates *exhaustiveDecomposition4* and *disjointDecomposition4* have been abbreviated to `exhDecomp4` and `disDecomp4` respectively, whereas the constant function symbol *animalAnatomicalStructure* has been abbreviated to `animalAS`.

The proof starts with the axioms involved (up to step 20). Note that E-Prover does not use all the axioms in the source file. Then, E-Prover assumes the negation of the objective (in step 21) and applies the initial transformation to all the formulae (up to step 84). From the transformation of the negated conjecture, E-Prover selects five disjuncts, which are those in steps 25-29. Note that, after negation, the variables `X1` and `X2` in the objective become existentially quantified and, thus, cause the introduction of the Skolem constant functions `esk1_0` and `esk2_0`, respectively. Besides, the disjuncts in steps 48, 53 and 71 come from the transformation of the formulae in 6, 7 and 9, where the disjuncts that are not going to be used in the proof have been discarded. The remaining formulae of the initial transformation are the disjuncts in steps 40, 74, 75, 81 and 84, which come directly from the formulae in steps 5, 10, 11, 17 and 20, respectively.

The inconsistency comes from the formulae `--instance(esk1_0,animal)` (in step 387) and `++instance(esk1_0,animal)` (in step 442). On one hand, the formula `--instance(esk1_0,animal)` results from a 4 step resolution sequence that starts with the formulae in steps 53 and 84 to finally yield `++disDecomp4(organism,animal,plant, microorganism)` (in step 215). Then, using the formula in step 71, we obtain `++disjoint(animal, plant)` (in step 340). Next, using the formula in step 48, we have that everything is not an instance of either `plant` or `animal`. Hence, since `++instance(esk1_0,plant)` (in step 26), we finally obtain the formula `--instance(esk1_0,animal)`. On the other hand, `++instance(esk1_0,animal)` results from an 8 step resolution sequence. First, from the formulae in steps 40, 75 and 81, it follows that everything is either an instance of `animalAS` or not an instance of `brain` (in step 157), and also that everything is either an instance of `organism` or not an instance of `plant` (in step 158). Then, using `++instance(esk2_0,brain)` (in step 27) and `++instance(esk1_0,plant)` (in step 26), we obtain, respectively, `++instance(esk2_0,animalAS)` (in step 239) and `++instance(esk1_0,organism)` (in step 249). Using these last two formulae and also `++part(esk2_0,esk1_0)`, `++instance(esk1_0,object` and `++instance(esk2_0,object)` (in steps 25, 28 and 29 respectively), we finally obtain `++instance(esk1_0,animal)` (in step 442) from the disjunct in step 74 (after resolution steps 183, 436, 439 and 440). Hence, we obtain a proof of the original goal.

C Translating SUMO into FOL-Formulae: A Detailed Example

In this section, we provide a detailed description of the translation of SUMO into FOL-formulae. Here, we especially focus on some details that have not been discussed in the body of the paper. We choose TPTP syntax to write FOL-formulae, since most of current FOL-theorem provers accept it. In this section, we just use the existential (?) and universal

(!) quantifiers and the classical connectives negation (∼), conjunction (&), disjunction (|), implication (=>) and equivalence (<=>). A whole description of TPTP syntax is available at `http://www.tptp.org`. In order to illustrate the translation of SUMO into FOL-formulae, we consider the following set of axioms. First, we take two of the main predefined predicates, which are $*instance* and $*subclass*:[24]

1. ($domain $instance 1 $object)
2. ($domain $instance 2 $class)
3. ($domain $subclass 1 $class)
4. ($domain $subclass 2 $class)
5. (**forall** (?X)
   ($subclass ?X ?X))
6. (**forall** (?X?Y?Z)
   (=>
   ($subclass ?X ?Y)
   ($subclass ?Y ?Z))
   ($subclass ?X ?Z))
7. (**forall** (?X?Y)
   (=>
   ($subclass ?X ?Y)
   ($subclass ?Y ?X))
   (equal ?X ?Y))
8. (**forall** (?X?Y?Z)
   (=>
   ($instance ?X ?Y)
   ($subclass ?Y ?X))
   ($instance ?X ?z))

Note that type information (axioms 1-4) is provided using the predefined predicate $*domain* and also the predefined constants $*object* and $*class*, which respectively correspond to the meta-level concepts of object and class. In our translation, meta-level type information is used during the translation for checking that definitions are consistent, but it does not directly appear in the resulting FOL-formulae. Predicates $*instance* and $*subclass* are defined as usual (axioms 5-8) and its translation into TPTP syntax is straightforward:

```
( ![X]: $subclass(X,X) )

( ![X,Y,Z]: ( $subclass(X,Y) & $subclass(Y,Z) )
          => $subclass(X,Z) )

( ![X,Y]: ( $subclass(X,Y) & $subclass(Y,X) )
```

[24]The prefix $- is used to denote predefined or distinguished constant/predicate names.

```
          => ( X = Y ) )

(![X,Y,Z]: ( $instance(X,Y) & $subclass(Y,Z) )
          => $instance(X,Z) )
```

In addition, we use some other predefined predicates, such as $\$partition$, $\$exhaustiveDecomposition$, $\$disjointDecomposition$ and $\$disjoint$, which are defined in the following way:

9. ($domain $partition 1 $class)
10. ($domain $partition 2 @$class)
11. (**forall** (?CLASS @ROW)
    (<=>
    ($partition ?CLASS @ROW)
    (=>
    ($exhaustiveDecomposition ?CLASS @ROW)
    ($disjointDecomposition ?CLASS ?ROW))))
12. ($domain $exhaustiveDecomposition 1 $class)
13. ($domain $exhaustiveDecomposition 2 @$class)
14. (**forall** (?CLASS @ROW)
    (<=>
    ($exhaustiveDecomposition ?CLASS @ROW)
    (**forall** (?X)
    (=>
    ($instance ?X ?CLASS)
    (@**or** (@ROW, ?HEAD, @_)
    ($instance ?X ?HEAD))))))
15. ($domain $disjointDecomposition 1 $class)
16. ($domain $disjointDecomposition 2 @$class)
17. (**forall** (?CLASS @ROW)
    (<=>
    (@**and** (@ROW, ?CLASS1, @TAIL)
    (@**and** (@TAIL, ?CLASS2, @_)
    ($disjoint ?CLASS1 ?CLASS2)))))
18. ($domain $disjoint 1 $class)
19. ($domain $disjoint 2 $class)
20. (**forall** (?CLASS1 ?CLASS2)
    (<=>
    ($disjoint?CLASS1?CLASS2)
    (**forall** (?INST)

```
(not
  (and
    ($instance?INST?CLASS1)
    ($instance?INST?CLASS2))))))
```

Here, the variable arity predicates $\$partition$, $\$exhaustiveDecomposition$ and $\$disjointDecomposition$ have been turned into binary predicates by means of the use of *row lists*, which are tuples of non-fixed (but finite) arity. Row lists allow us to group several elements in a single argument, which is marked with the prefix @- in type information (axioms 10, 13 and 16). Those arguments marked with @- are therefore always used with row variables/lists. Further, we also provide specific operators to deal with row variables/lists: the operators **@and** and **@or** , which take 3 variables as arguments (2 row variables and a single variable). These operators are a kind of iterators that enables us to iteratively process each element in the row variable/list. The first element (head) in the row variable/list that occurs in the first argument of a row operator can be addressed using the second argument of the row operator and the rest of elements (tail) are addressed in the third argument, which allows for recursive definitions. Thus, an **@and** (resp. **@or**) formula is translated into a conjunction (resp. disjunction) of formulas, one for each element in the row variable/list occurring in the first argument. For example, considering a row list of arity two, axioms 11, 14 and 17 are translated into:

```
( ![CLASS,ROW1,ROW2]:  $partition3(CLASS,ROW1,ROW2) <=>
      ( $exhaustiveDecomposition3(CLASS,ROW1,ROW2)
       & $disjointDecomposition3(CLASS,ROW1,ROW2) ) )

( ![CLASS,ROW1,ROW2]:  $exhaustiveDecomposition3(CLASS,ROW1,ROW2) <=>
      (![X]: $instance(X,CLASS) => ( $instance(X,ROW1)
                | $instance(X,ROW2) ) ) )

( ![CLASS,ROW1,ROW2]:  $disjointDecomposition3(CLASS,ROW1,ROW2) <=>
                $disjoint(ROW1,ROW2) )
```

Note that we use predicate arities as postfix in predicate names when combined with row variables/lists, since FOL-theorem provers restrict the use of predicate symbols to a single arity.

Finally, we are going to translate the following set of axioms

21. ($disjointDecomposition Relation @(BinaryRelation, TernaryRelation, QuaternaryRelation, QuintaryRelation, VariableArityRelation))
22. ($partition Relation 1 @(Predicate, Function, List))
23. ($partition Relation 1 @(TotalValuedRelation, PartialValuedRelation))
24. ($subclass BinaryRelation 1 Relation)
25. ($subclass ReflexiveRelation 1 BinaryRelation)
26. (<=>
      ($instance ?REL ReflexiveRelation)

(**forall** (?INST)
($holds3 ?REL ?INST ?INST)))

27. ($domain connected 1 Object)
28. ($domain connected 2 Object)
29. ($instance connected BinaryPredicate)
30. ($instance connected ReflexiveRelation)
31. ($subrelation meetsSpatially connected)
32. ($subrelation overlapsSpatially connected)
33. (=>
(connected ?OBJ1 ?OBJ2)
(**or**
(meetsSpatially ?OBJ1 ?OBJ2)
(overlapsSpatially ?OBJ1 ?OBJ2)))

where $*holds3* is the predicate used to write the reflexive property of binary relations in first-order logic (axiom 26). First, since the arities of the row lists in axioms 21, 22 and 23 are 5, 3 and 2 respectively, axioms 11, 14 and 17 have to be translated according to each arity. Then, axioms 21-23 are directly translated into:

```
$disjointDecomposition6(Relation,BinaryRelation,TernaryRelation
      QuaternaryRelation,QuintaryRelation,VariableArityRelation)

$partition4(Relation,Predicate,Function,List)

$partition3(Relation,TotalValuedRelation,PartialValuedRelation)
```

Then, the translation of axioms 24-26 is also direct:

```
$subclass(BinaryRelation,Relation)

$subclass(ReflexiveRelation,BinaryRelation)

( ![REL]: $instance(REL,ReflexiveRelation) <=>
        ![INST]: $holds3(REL,INST,INST) )
```

Next, we translate axioms 29, 30 and 33. Note that, since *meetsSpatially* and *overlapsSpatially* are subrelations of *connected*, type information for *meetsSpatially* and *overlapsSpatially* is inherited from axioms 27-28. Further, we use the constant *constConnected* for the occurrences of *connected* as term. Thus, we obtain:

```
$instance(constConnected,BinaryPredicate)

$instance(constConnected,ReflexiveRelation)

( ![OBJ1,OBJ2]:  ( $instance(OBJ2,Object) & $instance(OBJ1,Object) ) =>
   ( connected(OBJ1,OBJ2) => ( meetsSpatially(OBJ1,OBJ2) |
           overlapsSpatially(OBJ1,OBJ2) ) ) )
```

Finally, we add the reflection formula that allows to relate the predicates *connected* and $*holds3* via the constant *constConnected*:

```
( ![X,Y]: connected(X,Y) <=> $holds3(constConnected,X,Y) )
```

We also have to add a reflection formula for *meetsSpatially* and *overlapsSpatially* since those predicates are instance of *ReflexiveRelation*.

D Resources

In this section, we introduce the files contained in the Adimen-SUMO package,[25] which have been used along the work presented in this paper. The interested reader may try the examples described in this work, and many others, using a FOL-theorem prover such as E-Prover.

The source files that contain the ontology in KIF format are **predefinitions.kif**, **merge161.kif** and **milo187.kif**. The first file, **predefinitions.kif**, contains the definitions of the basic predicates that are used in the rest of the ontology, as described in Section 9. The syntax of this file also includes some non-KIF features, such as row operators (see Appendix C), that are very useful to support the translation. The whole description of these extra syntactic features is out of the scope of this paper (we plan to include a complete description in a future work), but its meaning can be easily inferred from the context. The last two files **merge161.kif** and **milo187.kif** correspond to the top and mid level of SUMO respectively. Some minor syntactic modifications have been done in these files in order to be adapted to our translator. Moreover, we have repaired all the axioms that produced an inconsistency.

The result of translating and eliminating inconsistencies from the above three files can be found in **adimen.sumo.eprover.tstp**, which has been prepared to be used with E-Prover.[26] This file has been tested using several FOL-theorem provers during many hours (even days) and no more inconsistencies have been found. Thus, it can be used to explore the reasoning capabilities of SUMO.

An example of inconsistency that has been found by an FOL-theorem prover in a preliminary version of **adimen.sumo.eprover.tstp** is described in Section 6 (see also Appendix A). The axioms that are necessary to reproduce the inconsistency have been collected in the file **disjoint.eprover.tstp**. The explanation of the inconsistency given by E-Prover can be consulted in **output.disjoint.eprover.txt**.

Regarding the reasoning capabilities of SUMO, in **brain.eprover.tstp** we have isolated the axioms from **adimen.sumo.eprover.tstp** that are necessary to infer that plants do not have brain (see Section 9 and Appendix B). The proof for the goal in **brain.eprover.tstp** given by E-Prover can be consulted in **output.brain.eprover.txt**.

[25] Available at `http://adimen.si.ehu.es/web/AdimenSUMO`.

[26] We also provide the file **adimen.sumo.vampire.tstp** that has been prepared to be used with the last versions of Vampire.

```
  1 : : ![X1]:![X2]:(disjoint(X1,X2)<=>((instance(X1,nonNullSet)&instance(X2,nonNullSet))&
              ![X3]:~((instance(X3,X1)&instance(X3,X2))))) : initial(‘‘disjoint.eprover.tstp’’, disjoint1)
  2 : : ![X4]:(instance(X4,abstract)<=>~(?[X5]:(located(X4,X5)|time(X4,X5)))) :
                                                    initial(‘‘disjoint.eprover.tstp’’, disjoint2)
  3 : : ![X6]:(instance(X6,physical)<=>?[X7]:?[X8]:(located(X6,X7)&time(X6,X8))) :
                                                    initial(‘‘disjoint.eprover.tstp’’, disjoint3)
  4 : : ![X9]:![X10]:![X11]:((instance(X9,X10)&subclass(X10,X11))=>instance(X9,X11)) :
                                                    initial(‘‘disjoint.eprover.tstp’’, disjoint4)
  5 : : subclass(nonNullSet,setOrClass) : initial(‘‘disjoint.eprover.tstp’’, disjoint5)
  6 : : subclass(region,object) : initial(‘‘disjoint.eprover.tstp’’, disjoint6)
  7 : : subclass(object,physical) : initial(‘‘disjoint.eprover.tstp’’, disjoint7)
  8 : : subclass(setOrClass,abstract) : initial(‘‘disjoint.eprover.tstp’’, disjoint8)
  9 : : disjoint(indoors,outdoors) : initial(‘‘disjoint.eprover.tstp’’, disjoint9)
 10 : : instance(outdoors,region) : initial(‘‘disjoint.eprover.tstp’’, disjoint10)
...
 18 : : [++instance(X2,nonNullSet),--disjoint(X1,X2)] : split_conjunct(15)          *** from (1)
...
 28 : : [--instance(X1,abstract),--located(X1,X2)] : split_conjunct(25)             *** from (2)
...
 35 : : [++located(X1,esk3_1(X1)),--instance(X1,physical)] : split_conjunct(33)     *** from (3)
...
 39 : : [++instance(X1,X2),--subclass(X3,X2),--instance(X1,X3)] : split_conjunct(38)  *** from (4)
...
 40 : : [++subclass(nonNullSet,setOrClass)] : split_conjunct(5)
 41 : : [++subclass(region,object)] : split_conjunct(6)
 42 : : [++subclass(object,physical)] : split_conjunct(7)
 43 : : [++subclass(setOrClass,abstract)] : split_conjunct(8)
 44 : : [++disjoint(indoors,outdoors)] : split_conjunct(9)
 45 : : [++instance(outdoors,region)] : split_conjunct(10)
...
 71 : : [++instance(outdoors,nonNullSet)] : spm(70,64)                  *** from (18, 44)
...
 78 : : [--instance(X1,abstract),--instance(X1,physical)] : spm(77,76)  *** from (28, 35)
...
 85 : : [++instance(X1,setOrClass),--instance(X1,nonNullSet)] : spm(84,66)  *** from (39, 40)
 86 : : [++instance(X1,abstract),--instance(X1,setOrClass)] : spm(84,67)    *** from (39, 43)
 87 : : [++instance(X1,object),--instance(X1,region)] : spm(84,68)          *** from (39, 41)
 88 : : [++instance(X1,physical),--instance(X1,object)] : spm(84,69)        *** from (39, 42)
...
130 : : [++instance(X1,abstract),--instance(X1,nonNullSet)] : spm(127,124)  *** from (86, 85)
...
154 : : [++instance(X1,physical),--instance(X1,region)] : spm(151,134)      *** from (88, 87)
...
164 : : [++instance(outdoors,physical)] : spm(163,65)                       *** from (154, 45)
...
172 : : [--instance(outdoors,abstract)] : spm(108,170)                      *** from (78, 164)
...
179 : : [--instance(outdoors,nonNullSet)] : spm(178,158)                    *** from (172, 130)
180 : : [--$true] : rw(179,118)                                             *** from (179, 71)
181 : : [] : cn(180)
182 : : [] : 181 : ‘proof’
```

Figure 3: An Inconsistency Discovered by E-Prover

```
  1 : conj : ![X1]:![X2]:((instance(X2,object)&instance(X1,object))=>~(((instance(X2,brain)&
             instance(X1,plant))&part(X2,X1)))) : initial(‘‘brain.eprover.tstp’’, goal)
  5 : : ![X3]:![X4]:![X5]:((instance(X3,X4)&subclass(X4,X5))=>instance(X3,X5)) :
                                                 initial(‘‘brain.eprover.tstp’’, predefinitionsB4)
  6 : : ![X6]:![X7]:(disjoint(X6,X7)<=>![X8]:~((instance(X8,X6)&instance(X8,X7)))) :
                                                 initial(‘‘brain.eprover.tstp’’, predefinitionsB5)
  7 : : ![X9]:![X10]:![X11]:![X12]:(partition4(X9,X10,X11,X12)<=>(exhDecomp4(X9,X10,X11,X12)&
             disDecomp4(X9,X10,X11,X12))) : initial(‘‘brain.eprover.tstp’’, predefinitionsB6)
  9 : : ![X9]:![X10]:![X11]:![X12]:(disDecomp4(X9,X10,X11,X12)<=>((disjoint(X10,X11)&
             disjoint(X10,X12))&disjoint(X11,X12))) : initial(‘‘brain.eprover.tstp’’, predefinitionsB8)
 10 : : ![X13]:![X14]:((instance(X13,object)&instance(X14,object))=>(((instance(X13,animalAS)&
             instance(X14,organism))&part(X13,X14))=>instance(X14,animal))) :
                                                 initial(‘‘brain.eprover.tstp’’, merge158B1)
 11 : : subclass(brain,animalAS) : initial(‘‘brain.eprover.tstp’’, milo184B1)
 17 : : subclass(plant,organism) : initial(‘‘brain.eprover.tstp’’, merge158B7)
 20 : : partition4(organism,animal,plant,microorganism) : initial(‘‘brain.eprover.tstp’’, merge158B10)
 21 : neg : ~(![X1]:![X2]:((instance(X2,object)&instance(X1,object))=>~(((instance(X2,brain)&
             instance(X1,plant))&part(X2,X1))))) : assume_negation(1)
...
 25 : neg : [++part(esk2_0,esk1_0)] : split_conjunct(24)                          *** from (21)
 26 : neg : [++instance(esk1_0,plant)] : split_conjunct(24)                       *** from (21)
 27 : neg : [++instance(esk2_0,brain)] : split_conjunct(24)                       *** from (21)
 28 : neg : [++instance(esk1_0,object)] : split_conjunct(24)                      *** from (21)
 29 : neg : [++instance(esk2_0,object)] : split_conjunct(24)                      *** from (21)
...
 40 : : [++instance(X1,X2),--subclass(X3,X2),--instance(X1,X3)] : split_conjunct(39)   *** from (5)
...
 48 : : [--disjoint(X1,X2),--instance(X3,X2),--instance(X3,X1)] : split_conjunct(45)   *** from (6)
...
 53 : : [++disDecomp4(X1,X2,X3,X4),--partition4(X1,X2,X3,X4)] : split_conjunct(51)     *** from (7)
...
 71 : : [++disjoint(X2,X3),--disDecomp4(X1,X2,X3,X4)] : split_conjunct(67)            *** from (9)
...
 74 : : [++instance(X1,animal),--part(X2,X1),--instance(X1,organism),--instance(X2,animalAS),
             --instance(X1,object),--instance(X2,object)] : split_conjunct(24)       *** from (10)
 75 : : [++subclass(brain,animalAS)] : split_conjunct(11)
 81 : : [++subclass(plant,organism)] : split_conjunct(17)
 84 : : [++partition4(organism,animal,plant,microorganism)] : split_conjunct(20)
...
157 : : [++instance(X1,animalAS),--instance(X1,brain)] : spm(156,125)                *** from (40, 75)
158 : : [++instance(X1,organism),--instance(X1,plant)] : spm(156,126)                *** from (40, 81)
...
183 : neg : [++instance(X1,animal),--part(esk2_0,X1),--instance(esk2_0,animalAS),
             --instance(X1,object),--instance(X1,organism)] : spm(181,122)           *** from (74, 29)
...
215 : : [++disDecomp4(organism,animal,plant,microorganism)] : spm(214,135)           *** from (53, 84)
...
239 : neg : [++instance(esk2_0,animalAS)] : spm(238,123)                             *** from (157, 27)
...
249 : neg : [++instance(esk1_0,organism)] : spm(248,121)                             *** from (158, 26)
...
340 : : [++disjoint(animal,plant)] : spm(180,337)                                    *** from (71, 215)
...
373 : : [--instance(X1,plant),--instance(X1,animal)] : spm(176,372)                  *** from (48, 340)
...
387 : neg : [--instance(esk1_0,animal)] : spm(386,121)                               *** from (373, 26)
...
436 : neg : [++instance(X1,animal),--part(esk2_0,X1),--$true,--instance(X1,object),
             --instance(X1,organism)] : rw(187,247)                                  *** from (183, 239)
...
439 : neg : [++instance(esk1_0,animal),--instance(esk1_0,object),--instance(esk1_0,organism)] :
                                                 spm(438,124)                        *** from (436, 25)
440 : neg : [++instance(esk1_0,animal),--$true,--instance(esk1_0,organism)] : rw(439,120)  *** from (439, 28)
441 : neg : [++instance(esk1_0,animal),--$true,--$true] : rw(440,257)                 *** from (440, 249)
442 : neg : [++instance(esk1_0,animal)] : cn(441)
443 : neg : [] : sr(442,395)                                                         *** from (442, 387)
444 : neg : [] : 443 : ’proof’
```

Figure 4: Do Plants Have Brain?