Adimen-SUMO: Reengineering an Ontology for First-Order Reasoning

Javier Álvez *
javier.alvez@ehu.es
Department of Languages and Information Systems
University of the Basque Country

Paqui Lucio
paqui.lucio@ehu.es
Department of Languages and Information Systems
University of the Basque Country

German Rigau
german.rigau@ehu.es
Department of Languages and Information Systems
University of the Basque Country

In this paper, we present Adimen-SUMO, an operational ontology to be used by first-order theorem provers in intelligent systems that require sophisticated reasoning capabilities (e.g. Natural Language Processing, Knowledge Engineering, Semantic Web infrastructure, etc.). Adimen-SUMO has been obtained by automatically translating around 88% of the original axioms of SUMO (Suggested Upper Merged Ontology). Our main interest is to present in a practical way the advantages of using first-order theorem provers during the design and development of first-order ontologies. First-order theorem provers are applied as inference engines for reengineering a large and complex ontology in order to allow for formal reasoning. In particular, our study focuses on providing first-order reasoning support to SUMO. During the process, we detect, explain and repair several important design flaws and problems of the SUMO axiomatization. As a by-product, we also provide general design decisions and good practices for creating operational first-order ontologies of any kind.

Keywords: Ontologies, First-order Logic, Automated Reasoning, Knowledge Engineering, Linked Open Data

Introduction

Recently, the Semantic Web community has become very interested in having practical inference engines for ontological reasoning (Chandrasekaran, Josephson, & Benjamins, 1999; Noy & McGuinness, 2001; Staab & Studer, 2009; d'Aquin & Noy, 2012). 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 formally defined. Automated reasoning uses mechanical procedures to obtain a large body of deducible knowledge that can be inferred from a compact modelling.[1]

A 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 (Gruber, 2009). It is also well-known that the expressive power of the underlying logic jeopardizes the computational tractability and the inherent complexity of logical reasoning. Indeed, very expressive logics are undecidable or semi-decidable with high worst-case complexity. Beyond undecidability, a highly expressive logic language may be incomplete, i.e. there is no finite proof system that will prove all entailed sentences in the logic. As a consequence, the trade-off between expressiveness and reasoning efficiency (even decidability) is a key point for the design of formal ontologies.

Today, the family of Web Ontology Languages, including OWL-DL (Horrocks & Patel-Schneider, 2004), is the most common formal knowledge representation formalism, being accepted and standardized by the W3C (World Wide Web Consortium). OWL-DL is a powerful knowledge representation formalism with high computational efficiency and theoretically founded in *description logics* (DL), which can be embedded into fragments of first-order logic. Additionally, pure DLs are subsets of the *guarded fragment* (GF) of first-order logic, defined through the relativisation of quantifiers by atomic formulas. The guarded fragment was introduced in (Andréka, Németi, & van Benthem, 1998), where the authors prove that the satisfiability problem for GF is decidable. Indeed, OWL-DL reasoners, such as Pellet (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007) or Fact++ (Tsarkov & Horrocks, 2006), implement very efficient decision algorithms for OWL-DL theories (Motik, Shearer, & Horrocks, 2009).

[1] Formally, deduction is the logical action of obtaining statements that are true in all models of an axiomatization.

However, OWL-DL decidability is achieved at the price of losing expressiveness. Thus, state-of-the-art OWL-DL reasoners are unable to cope with more expressive ontologies such as Cyc (Matuszek, Cabral, Witbrock, & DeOliveira, 2006), DOLCE (Gangemi, Guarino, Masolo, Oltramari, & Schneider, 2002) or SUMO (Niles & Pease, 2001b).

Likewise, first-order logic (FOL) is a very well-known and expressive formalism, although reasoning in FOL is undecidable. Lately, impressive progress has been made in first-order (FO) automated reasoning. Every year, the CASC competition[2] (Pelletier, Sutcliffe, & Suttner, 2002; Sutcliffe & Suttner, 2006) evaluates the performance of sound, fully automatic, classical FO automated theorem provers (ATP) on a bounded number of eligible problems, chosen from the TPTP Problem Library[3] (Sutcliffe, 2009). As a result, there exists an important collection of FO theorem provers, such as Vampire (Riazanov & Voronkov, 2002) and E-Prover (Schulz, 2002).

Some of these FO theorem provers have been used for reasoning with expressive ontologies such as SUMO (Suggested Upper Merged Ontology).[4] In particular, the authors of (Horrocks & Voronkov, 2006) give many interesting hints for adapting the existing general purpose FO theorem provers for checking its consistency. The authors also provide two examples of inconsistencies that were detected in SUMO. Additionally, in (Pease & Sutcliffe, 2007), the authors report some preliminary experimental results evaluating the query timeout for different options when translating SUMO into FOL.[5] Evolved versions of the translation described in (Pease & Sutcliffe, 2007) can be found in the TPTP Library. In the sequel, we refer to this translation as TPTP-SUMO.

In this paper, we present Adimen-SUMO,[6] an operational off-the-shelf FO version of SUMO. Our main interest is to present in a practical way the advantages of using FO theorem provers as inference engines for working with large and complex ontologies. In particular, we have concentrated our efforts on studying, revising and improving SUMO, although a similar approach could be applied to Cyc, DOLCE or any other expressive ontology. We decided to focus on SUMO since we are aware of its intrinsic problems, and also of the fact that its design and development have been harshly criticized in the community of ontology developers and users from the very beginning. For example, in (Oberle et al., 2007) the authors describe some of the shortcomings that the axiomatization of SUMO suffers from and propose the integration of DOLCE and SUMO.

Initially, our intention was to use existing FO theorem provers as inference engines for working with the ontology. For this purpose, we started selecting the subset of FOL axioms from SUMO. These axioms only required a syntactic transformation to be used by FO theorem provers. Next, incrementally, we tried several options to transform the rest of SUMO into FOL. The initial inconsistencies detected by the FO theorem provers led us to start debugging the ontology. On the basis of the explanations (or refutations) provided by the FO theorem provers, we investigated the conflicting axioms. Once we determined the final reason of the inconsistencies, we tested possible solutions for the involved axioms. Following this procedure, we found spurious 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. Furthermore, we also discovered basic design problems in SUMO which impede its appropriate use by a FO theorem prover. Thus, we reengineered the ontology to solve these basic design problems.

Figure 1. The *brain-plant* Example

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

As a result of this process, we have developed an automatic translator that transforms around 88% of the original SUMO axioms into FOL. Our translated ontology, which is called Adimen-SUMO, can be used by FO theorem provers to perform formal reasoning about the properties and relations of the classes defined by the ontology. For example, it is now very straightforward to infer from Adimen-SUMO that plants do not have brain (or any *AnimalAnatomicalStructure*). In fact, this question cannot be solved using the TPTP-SUMO translation. To answer this question, it is enough to prove that the goal in Figure 1 follows from Adimen-SUMO.[7] In Section *"Adimen-SUMO"*, we provide additional examples of inferences obtained by FO theorem provers using Adimen-SUMO that cannot be obtained using any other version of SUMO, including TPTP-SUMO. This type of non-trivial inferences could be very useful for a wide range of knowledge intensive applications. For instance, to help validating the consistency of associated semantic resources like WordNet[8] (Fellbaum, 1998; Niles & Pease, 2003), or to derive new explicit knowledge from them.

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

[3] http://www.tptp.org

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

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

[6] In Basque, Adimen means intelligent.

[7] In Appendix B, we provide a full proof that is obtained with a FO theorem prover.

[8] http://wordnet.princeton.edu

Figure 2. Partial knowledge about plants represented in WordNet version 3.0.

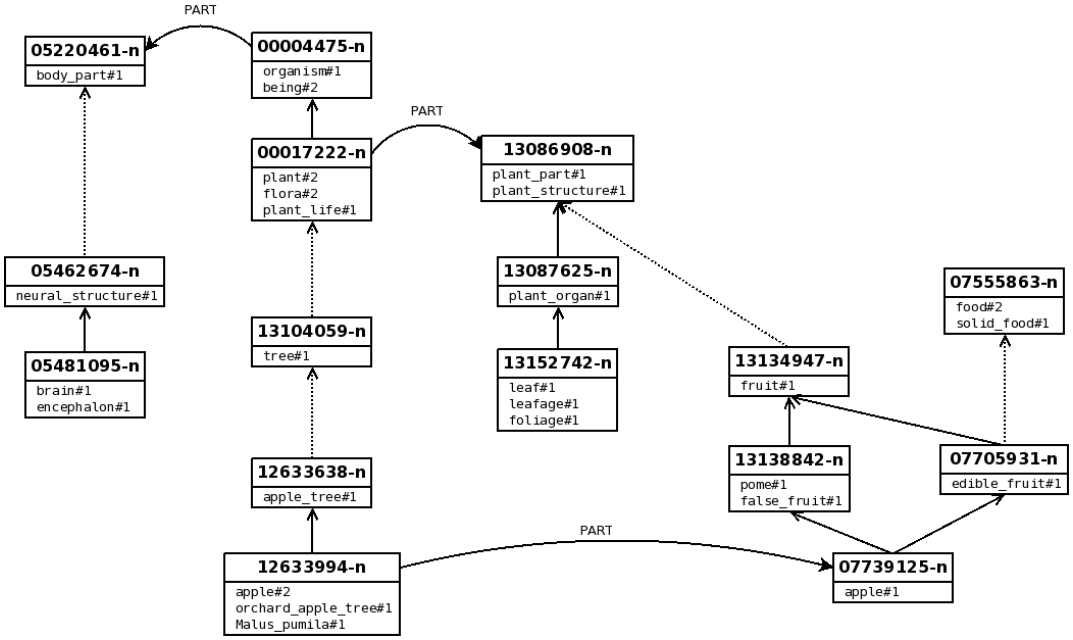

Figure 2 shows a partial view of the knowledge contained in WordNet. In this figure, each box represents a synset (set of synonyms) and includes its word senses and offset number with part-of-speech,[9] and solid arrows represent immediate relations (subclass-of[10] or part[11]) whereas dot arrows represent longer subclass relation paths. Using WordNet, a program can establish that an *apple (apple^2_n, the tree*) has *apple (apple^1_n, the fruit)* and using a very simple inference mechanism that an *apple (the tree)* also has *fruit* and possibly *food*. However, only considering the current knowledge represented in WordNet, it is impossible to establish if an *apple (the tree)* has a *leaf* or not. For instance, a systematic propagation procedure would also infer that an *apple (the tree)* also has a *brain*. Now, using Adimen-SUMO and its tight connection to WordNet,[12] we can formally establish that no *plant* has *brain* as well as a *plant* can have a *leaf*. Obviously, similar reasoning mechanisms can also be established for other semantic properties and relations among WordNet concepts (Harabagiu & Moldovan, 1998; Álvez et al., 2008; Verdezoto & Vieu, 2011) or other semantic resources connected to SUMO. Moreover, WordNet is being used world-wide to anchor different wordnets in other languages.[13] Therefore, similar inferences can be obtained for languages other than English. Likewise, WordNet is connected to several databases such as OpenCyc (Reed & Lenat, 2002), DBpedia (Auer et al., 2007; Bizer, Lehmann, et al., 2009) or YAGO (Hoffart, Suchanek, Berberich, & Weikum, 2013) thanks to the Linked Open Data (LOD) cloud initiative (Bizer, Heath, & Berners-Lee, 2009). The interlinking of these diverse databases to a fully operational upper level ontology promises a "Web of Data" that will enable machines to more easily exploit its content (Jain, Hitzler, Yeh, Verma, & Sheth, 2010). Thus, the use of Adimen-SUMO and its tight connection to WordNet can significantly benefit both the Artificial Intelligence and Semantic Web communities by providing better reasoning capabilities to a large set of resources and databases integrated into the LOD cloud.

The outline of the paper is as follows. In the next section, we introduce SUMO, the ontology of interest to this work. Section *"Logic Languages and KIF"* is focused on the format used for describing SUMO and its expressiveness. In Section *"First-Order Theorem Proving"*, we describe the operation of FO theorem provers, and in Section *"Detecting Inconsistencies"* we illustrate the use of theorem provers for finding inconsistencies in an ontology in an automatic way. Section *"Translating SUMO into First-Order Logic"* is devoted to the process of translating and reengineering SUMO into a FOL ontology, and in Section *"Adimen-SUMO"* we present the ontology that results from our work. In Section *"Concluding Remarks"*, we summarize our main contributions and discuss related work. Finally, for the interested reader, we provide four appendixes. Appendix

[9] Instead of offset numbers, we will use the following format to refer to a particular word sense: word^s_p, where p is the part-of-speech (n for nouns) and s is the sense number.

[10] In WordNet, hyponymy relations

[11] In WordNet, meronymy relations

[12] *brain*^1_n is subsumed by the SUMO concept *Brain* and *plant*^2_n is equivalent to the SUMO concept *Plant*.

[13] `http://www.globalwordnet.org/`

A gives a detailed example of how to analyze the traces obtained from FO theorem provers in order to debug the ontology and Appendix B presents another detailed example of the new reasoning capabilities provided by Adimen-SUMO. Appendix C provides a thorough description of our translation of SUMO into FOL. The Adimen-SUMO package[14] is described in Appendix D.

Suggested Upper Merged Ontology

An upper ontology is limited to concepts, relations and axioms that are generic or abstract. 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.

SUMO[15] (Niles & Pease, 2001b) has its origins in the nineties, when a group of engineers from the IEEE Standard Upper Ontology Working Group pushed for a formal ontology standard. Their goal was to develop a standard upper ontology to promote data interoperability, information search and retrieval, automated inference and natural language processing. After a long and troublesome discussion, the moderators decided to pick some available upper ontologies (e.g. John Sowa's upper ontology (Sowa, 2000), James Allen's temporal axioms (Allen, 1984), Nicola Guarino's formal mereotopology (Borgo, Guarino, & Masolo, 1996, 1997), etc.) and merged them into SUMO (Niles & Pease, 2001a). The merging was severely criticized because of the lack of criteria and strategy for the merging, and the lack of formal procedure for checking the results.

Currently, SUMO (including the extended set of domain ontologies based on it) consists of about 20,000 terms and about 70,000 axioms. However, in the work reported here, we concentrate on the upper part of the ontology. That is, on SUMO itself (file Merge.kif, version 1.78) and the mid-level ontology (file Mid-level-ontology.kif, version 1.114), which consists of about 1,000 terms and 4,000 axioms. The exact number of axioms in each part depends on the particular version.[16]

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 (Niles & Pease, 2003). WordNet[17] (Fellbaum, 1998) is by far the most widely-used lexical 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, ⟨*brain*, *encephalon*⟩ 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 "that part of the central nervous system that includes all the higher nervous centers; enclosed within the skull; continuous with the spinal cord", 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. The WordNet-SUMO mapping provides three types of connections: equivalent mapping, subsuming mapping and instance mapping. For example, the synset ⟨*brain*, *encephalon*⟩ is subsumed by the SUMO concept *Brain*.

In turn, WordNet is being used world-wide to anchor different types of semantic knowledge. For instance, the Multilingual Central Repository (MCR)[18] (Atserias, Rigau, & Villarejo, 2004; Gonzalez-Agirre, Laparra, & Rigau, 2012b, 2012a) integrates in the same EuroWordNet multilingual framework (Vossen, 1998) wordnets from five different languages: English, Spanish, Catalan, Basque and Galician together with domain knowledge (Magnini & Cavaglià, 2000; Bentivogli, Forner, Magnini, & Pianta, 2004) and ontologies like the EuroWordNet Top Ontology (Álvez et al., 2008) and SUMO (Pease & Fellbaum, 2010).

Furthermore, SUMO has also been merged with YAGO [19] (Suchanek, Kasneci, & Weikum, 2007; Hoffart et al., 2011), thus combining the rich axiomatization of SUMO with the large number of individuals acquired from Wikipedia and GeoNames. Actually, as part of the Linking Open Data project (Bizer, Heath, & Berners-Lee, 2009), YAGO is also integrated into DBpedia (Kobilarov, Bizer, Auer, & Lehmann, 2009), 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 (Jain, Hitzler, Sheth, Verma, & Yeh, 2010).

Additionally, the SMO category in the LTB division – FO non-propositional theorems (axioms with a provable conjecture) from Large Theories presented in Batches – of the CADE ATP System Competition CASC (Pelletier et al., 2002; Sutcliffe & Suttner, 2006) is based on problems taken from SUMO.

Logic Languages and KIF

SUMO is expressed in SUO-KIF (Standard Upper Ontology Knowledge Interchange Format, see (Pease, 2009)),

[14] `http://adimen.si.ehu.es/web/AdimenSUMO`.

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

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

[17] `http://wordnet.princeton.edu`

[18] `http://adimen.si.ehu.es/web/MCR`

[19] `http://www.mpi-inf.mpg.de/yago-naga/yago`

which is a dialect of KIF (Knowledge Interchange Format, see (Genesereth et al., 1992)). Both KIF and SUO-KIF are knowledge interchange formats that provide a logic-based notation for a suitable representation of knowledge. KIF and SUO-KIF are not exactly formal languages (such as the FO language), nor executable languages (such as Prolog). In fact, they can be used to write FO formulas, but its syntax goes beyond FOL. For instance, they can also be used to write second-order (SO) formulas, allowing for variables representing predicates and quantification over them. Moreover, KIF allows higher-order predicates – that is, predicates having other predicates as arguments – and even formulas acting as arguments of predicates. Another non-FO feature of KIF are the so-called *row variables*. These variables are related to infinitary logic (Keisler, 1971). Row variables allow the use of predicates and functions of arbitrary arity. A formal declarative semantics for the first-order sublanguage SKIF of KIF is presented in (Hayes & Menzel, 2001). However, some constructors of KIF and SUO-KIF lack corresponding model-theoretic constructions which would give them a rigorous meaning (Menzel & Hayes, 2003). For simplicity, from now on we refer to SUO-KIF as KIF.

In KIF syntax, the operators are written in prefix notation and, additionally, conjunction and disjunction are *n*-ary. The only naming convention is that variable names always start with a question mark '?', since there is no restriction about predicate and function names. For a complete reference to KIF, the interested reader is referred to (Genesereth et al., 1992). With respect to FO formulas, we use the standard notation with the following notational conventions:

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

Additionally, and as long as it is possible, we always try to use the same function and predicate names (starting with lower case in the case of predicate names) used in SUMO.

KIF can be used to express FO formulas. For example, the following KIF expression

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

corresponds to the FO formula

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

where

- C is a constant (0-ary function symbol),
- p and q are predicate symbols,
- x and y are variables.

In the rest of this section and by means of some examples, we discuss the expressiveness of KIF in comparison with OWL-DL, FOL and higher-order logic (HOL) languages.

First, some SUMO axioms cannot be expressed in OWL-DL due to its syntactic restrictions.[20] For example, the arity of predicates in OWL-DL is restricted to be at most two. As a consequence the following SUMO axiom

```
(=>
   (occupiesPosition ?PERSON ?POSITION
                                    ?ORGANIZATION)
   (member ?PERSON ?ORGANIZATION))
```

cannot be stated using OWL-DL because *occupiesPosition* is a ternary predicate. Moreover, the OWL-DL restrictions on the use of binary predicates prevents not only to replace the above ternary predicate with two binary predicates, but also to write *transitivity-like* axioms.[21] This is the case of e.g. the following SUMO axiom:

```
(=>
   (and
      (hole ?HOLE ?OBJ_1)
      (part ?OBJ_1 ?OBJ_2))
   (or
      (overlapsSpatially ?HOLE ?OBJ_2)
      (hole ?HOLE ?OBJ_2)))
```

The latter two axioms are pure FO formulas. However, there are many ways in which KIF goes beyond FOL. Our proposal deals with some of them, but avoids some others. For instance, the following SUMO axiom

```
(<=>
   (instance ?REL SymmetricRelation)
   (forall (?INST_1 ?INST_2)
      (=>
         (?REL ?INST_1 ?INST_2)
         (?REL ?INST_2 ?INST_1))))
```

does not correspond to any FO 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. Moreover, axioms are considered to be universally closed, which in this concrete case means that the KIF axiom can be considered to be prefixed by **forall** (?REL). The above kind of formula is not allowed in FOL, but it is permitted in second-order logic (SOL). In Section *"Translating SUMO into First-Order Logic"*, we explain a relaxed interpretation of this SO feature which allows us to translate many KIF axioms into FO formulas.

The row variables in KIF serve to express properties of relations of any arbitrary arity. This feature of KIF, when used

[20] Although SUMO axioms could be expressed in more expressive sublanguages of OWL, like OWL-Full.

[21] For instance, when describing a relation between two objects provided that each of both is related with a third object.

in its whole expressiveness, relates KIF to infinitary logic (Keisler, 1971). 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))))
```

However, the above axiom does not determine the arity of ?REL. In Section *"Translating SUMO into First-Order Logic"*, we explain how to translate axioms with row variables into FO formulas.

KIF syntax also allows formulas to be *embedded* as arguments of predicates, which is not possible in the FO language. For example, in the following SUMO 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 kind of embedded formula enables to write expressions in the style of modal and BDI[22] logics (as introduced in (Cohen & Levesque, 1990)). An example of BDI formula is

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

where the formula (possesses ?AGENT ?OBJ) is an argument of the predicate *desires*. However, without a suitable translation, these axioms are syntactically unacceptable by any FO theorem prover. We currently do not consider those axioms in SUMO that involve embedded formulas. In general, the SUMO axioms that our translation avoids can be considered *higher-order logic* (HOL). More precisely, most of these axioms correspond to different types of first-order modal logics – e.g. temporal, BDI, etc. – that can be encoded in the more expressive HOL. Moreover, since higher-order, modal and temporal logics are conservative extensions (or super-logics) of FOL, there is no theoretical problem to adapt the translation to a more expressive (but conservative) logic. Actually, our proposal for partially translating SUMO could be extended in the future to either a HOL-based or a FOL-based translation in order to deal with HO axioms. In particular, for the latter case there are well-known translations of some concrete temporal logics into FOL (Abadi, 1989) which, in future work, can be taken into account to translate temporal axioms. Indeed, automated reasoning in HOL and non-classical logics is rapidly evolving. In particular, there are very recent efforts to foster the HO reasoning development through the TPTP infrastructure (Sutcliffe & Benzmüller, 2010) and also on the application of HO automated theorem provers to ontological reasoning (Benzmüller & Pease, 2012). However, the development and optimization of FO theorem provers has reached a higher degree of progress in the last decades and, as a result, the scientific community has at its disposal several FO theorem provers that are able to deal with large inputs (such as a large part of SUMO) in a reasonable way. Additionally, a large part (even the whole) of many ontologies (including SUMO) can be represented in FOL. Consequently, we decided to concentrate on the translation of a large part of SUMO to FOL.

First-Order Theorem Proving

In this section, we briefly review the most important aspects of FO automated reasoning and the main theorem provers that we used in our work.

Automated theorem proving is one of the most well-developed subfields of automated reasoning. Extensive work has been done in order to develop techniques and tools for automatically deciding whether a set of axioms is satisfiable or if a formula can be derived from a set of axioms. Depending on the underlying logic, the satisfiability problem can be decidable, semi-decidable or undecidable. Within automated theorem proving, FO theorem proving is one of the most mature subfields and there are many fully automated systems that implement different techniques for FO 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, on the assumption 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 effectively proving that there is no inconsistency in Φ. Another technique consists in directly building a model for $\Phi \wedge \neg\phi$. In this case, if there is no model, then we can decide that $\Phi \wedge \neg\phi$ 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, refutation-based techniques are able to find a refutation when it does exist. Otherwise, when there is no refutation, the system may never terminate (i.e. it does not answer) if the refutation search space is infinite. Addi-

[22] Belief, Desires and Intentions

tionally, a model for a satisfiable set of axioms may also be infinite. Note that the construction of an infinite model that is not finitely representable requires an infinite process.

There is a large library of standard benchmark examples – the *Thousands of Problems for Theorem Provers (TPTP)* (Sutcliffe, 2009) – 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)* (Pelletier et al., 2002; Sutcliffe & Suttner, 2006), which evaluates the performance of FO theorem provers. In this competition, some of the most successful systems are Vampire (Riazanov & Voronkov, 2002) and E-Prover (Schulz, 2002) (which are refutation-based automated theorem provers), and Darwin (Baumgartner, Fuchs, & Tinelli, 2006), Paradox (Claessen & Sörensson, 2003) and iProver (Korovin, 2008) (which try to find a model for the given theory). Some other systems, such as MetaProver (Streeter, Golovin, & Smith, 2007) and SinE[23] (Hoder & Voronkov, 2011) (the winning system of the SMO category in the 2009 edition of CASC and integrated in the last versions of Vampire), are based on the above systems, mainly Vampire and E-Prover, but implement different resolution strategies, especially related to axiom selection. Since them, many other systems have also adopted similar strategies. Apart from the CASC competition, it is worth mentioning the Mace4 (that searches for finite models of first-order formula) and Prover9 (refutation-based) systems, which are the successors of Mace2 (McCune, 2001) and Otter (McCune & Wos, 1997), respectively.

In our research work, we have tested (and used) all the above-mentioned systems, mainly the Vampire and E-Prover systems. We have translated a large part of SUMO into a set of FO axioms, which is the FO ontology used as input for a theorem prover. In general, we have two options when running a FO theorem prover regarding execution time. According to the first option, the user sets no time limit. Hence, if the ontology is satisfiable, then the system may not terminate. According to the second option, we specify 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, which helps us debugging the ontology. On the contrary, if the theorem prover finds a model, then we have a proof that the ontology is consistent. Otherwise (no proof and no model), we receive no information from the theorem prover. For our purposes, we expect the theorem prover to find inconsistencies until no more inconsistency arises. However, finding a model is a hard task for current theorem provers. Thus, proving the consistency of the ontology is out of the scope of this work. In this sense, the work presented in (Kutz & Mossakowski, 2011) provides a plausible natural direction for future work in order to provide a consistency proof for Adimen-SUMO.

Detecting Inconsistencies

A FO theorem prover allows to detect inconsistencies in FO ontologies and, furthermore, to analyse the trace of the inconsistency proof (refutation) to discover the axioms that cause the inconsistency. In this section, we illustrate this process and provide some examples. In these examples, we point out some SUMO axioms that were in conflict and the solution we adopted to avoid exactly the mentioned conflict. In this sense, we should warn that the solutions described in this section are still preliminary and that further transformations are required, as described in Section *"Translating SUMO into First-Order Logic"*, in order to avoid other non-desirable inferences (even inconsistencies).

The procedure we have used for finding inconsistencies in the ontology can be sketched as follows:

1. An automated procedure translates a large part of SUMO (and discharges the remaining axioms).
2. Then, the whole resulting formula is given (as input) to a theorem prover for automatically finding an inconsistency (that is, without providing a goal).
3. When a refutation is found, the theorem prover provides a description of the proof, from which we select the collection of axioms involved in that refutation.
4. With the help of theorem provers (for example, for finding minimal inconsistent subcollections of axioms), we identify the source of the inconsistency and repair it.
5. Once we repair the problem, the process is repeated from the beginning.

In the following subsections, we describe two types of inconsistencies that require different reasoning capabilities to detect and correct them. Firstly, in Subsection *"Simple Inconsistencies"*, we describe a kind of inconsistency that could also be detected using more simple tools than FO theorem provers and that are simple to correct. Secondly, finding the kind of inconsistency described in Subsection *"Complex Inconsistencies"* requires FO theorem proving, and correcting the involved axioms is not trivial since it directly affects some design decisions.

Simple Inconsistencies

The first kind of inconsistency corresponds to incompatibilities in the structure of classes of SUMO. Instead of using FO theorem provers, more efficient tools and reasoners can be used for detecting those inconsistencies, as it is done for example in Protégé[24] with the automatic reasoners Fact++ (Tsarkov & Horrocks, 2006) and Pellet (Sirin et al., 2007).

In our case, the following inconsistency was detected using FO theorem provers by just keeping the facts involving *disjoint*, *subclass* and *instance* predicates, together with the common axiomatization of *disjoint*, *subclass* (reflexivity,

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

[24] `http://protege.stanford.edu/`

Figure 3. An Example of Complex Inconsistency.

#	KIF Axioms	FO Formulas
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$_2$))))))	$\forall x_1 \forall x_2$ ($disjoint(x_1, x_2) \leftrightarrow$ ($instance(x_1, NonNullSet) \wedge$ $instance(x_2, NonNullSet) \wedge$ $\forall y \neg[\ instance(y, x_1) \ \wedge \ instance(y, x_2)\ ]$))
2.	(<=> (instance ?ABS Abstract) (**not** (**exists** (?POINT) (**or** (located ?ABS ?POINT) (time ?ABS ?POINT)))))	$\forall x$ ($instance(x, Abstract) \leftrightarrow \neg\exists y$ ($located(x, y) \vee$ $time(x, y)$))
3.	(<=> (instance ?PHYS Physical) (**exists** (?LOC ?TIME) (**and** (located ?PHYS ?LOC) (time ?PHYS ?TIME))))	$\forall x$ ($instance(x, Physical) \leftrightarrow \exists y_1 \exists y_2$ ($located(x, y_1) \wedge$ $time(x, y_2)$))
4.	(=> (**and** (subclass ?X ?Y) (instance ?Z ?X)) (instance ?Z ?Y))	$\forall x \forall y \forall z$ (($subclass(x, y) \ \wedge \ instance(z, x)$) $\rightarrow$ $instance(z, y)$)
5.	(subclass NonNullSet SetOrClass)	$subclass(NonNullSet, SetOrClass)$
6.	(subclass Region Object)	$subclass(Region, Object)$
7.	(subclass Object Physical)	$subclass(Object, Physical)$
8.	(subclass SetOrClass Abstract)	$subclass(SetOrClass, Abstract)$
9.	(disjoint Indoors Outdoors)	$disjoint(Indoors, Outdoors)$
10.	(instance Outdoors Region)	$instance(Outdoors, Region)$

transitivity) and *instance* (inheritance through *subclass*). The axioms in conflict are:[25]

(instance Gray SecondaryColor)

(instance Gray SystemeInternationalUnit)

According to SUMO knowledge, the class *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. Further, the classes *Attribute* and *Quantity* are inferred to be disjoint. Henceforth, the inconsistency is detected, because *Gray* cannot be a common instance of two disjoint classes.

After advising about this inconsistency to the developers of SUMO, it was corrected in Mid-level-ontology.kif version 1.44 by replacing the above first axiom with

(instance GrayColor SecondaryColor).

Complex Inconsistencies

An example of more substantial bugs that can be found using theorem provers is given in Figure 3.[26] From that set of axioms, a theorem prover can infer falsehood or inconsistency. Analysing the proof provided by the theorem prover (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, it is easy to see that formula d is the negation of formula g. These two formulas are obtained from Axioms 2 and 3 respectively since *Outdoors* is an instance of both *Abstract* and *Physical*: on one hand, *Outdoors* is an instance of *Physical* because it is also an instance of *Region*; on the

[25] Extracted from Merge.kif version 1.36.

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

Figure 4. Translation of First-Order Axioms.

KIF Axiom	FO Formula
(=> (immediateInstance ?ENTITY ?CLASS) (**not** (**exists** (?SUBCLASS) (**and** (subclass ?SUBCLASS ?CLASS) (**not** (equal ?SUBCLASS ?CLASS)) (instance ?ENTITY ?SUBCLASS)))))	$\forall x_1 \forall x_2$ ($immediateInstance(x_1, x_2) \rightarrow$ $\neg\exists y$ ($subclass(y, x_2) \wedge \neg(y = x_2) \wedge instance(x_1, y)$))

other hand, *Outdoors* is an instance of *Abstract* because it is also an instance of *NonNullSet*.

In set theory, the empty set is disjoint from any other set. Hence, we decided to replace Axiom 1 with the following one

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

as suggested to the developers of SUMO and introduced in Merge.kif version 1.22. This replacement repairs the inconsistency. [27]

Translating SUMO into First-Order Logic

In this section, we introduce the main contribution of this work, which is a translation of a large part of SUMO into a FOL formula. The translation is explained in four steps, each of them focusing on a different kind of axiom. Additionally, we compare our solutions for each case with the ones proposed by other works that can be found in the literature. In particular, with the proposal in (Pease & Sutcliffe, 2007) – which partially includes some of our suggestions to the authors – and with TPTP-SUMO.

We describe our translation of SUMO in the following four subsections. In the first subsection, we describe the translation of first-order axioms, which is the simplest case. In the second subsection, we explain the translation of second-order axioms. The translation of axioms containing row variables is described in the third subsection. Finally, in the last subsection, we explain a suitable translation of the type information contained in SUMO, which also requires some additional transformation of the information in the ontology. We have developed a program which implements the four translation steps. The result of this automatic translation process is Adimen-SUMO, which is described in Section *"Adimen-SUMO"*.

First-Order Axioms

The translation of first-order axioms in SUMO is straightforward since it simply requires a syntax transformation. In Figure 4, we illustrate the transformation by means of the axiom that characterizes the concept of *immediateInstance*.

In SUMO, around 71% (5580 of 7787) of all axioms can be translated in this direct way.

Second-Order Axioms

The *standard semantics* of SOL interprets an n-ary predicate symbol by any possible subset of n-tuples of values of the discourse domain. Under this standard semantics, SOL is incomplete. That is, there is no deductive calculus able to derive all theorems. Since FOL has a complete calculus,[28] it is obvious that SOL, under standard semantics, cannot be translated into FOL. However, there is a well-known translation of SOL into FOL (see (Manzano, 1996) for a detailed explanation of this translation) which preserves a non-standard semantics – called Henkin semantics (Henkin, 1950) – 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 SO formulas into FO ones is to use a collection of predicates $holds_k$, where $k \geq 2$ stands for the arity of the predicate.[29] Using these predicates, any atom $P(t_1, \ldots, t_n)$ where P is a variable is translated into $holds_{n+1}(P, t_1, \ldots, t_n)$. However, in order to preserve Henkin semantics, an infinite collection of FO axioms, called *comprehension axioms*, should be added to the axiomatization. Roughly speaking, comprehension axioms characterize the predicate $holds_k$. Of course, this is a great limitation for automated reasoning.

However, ontologies are finite theories over a finite alphabet and the intended meaning of a second-order KIF axiom

[27] Axiom 10 has been replaced with (subclass Outdoors Region) in the latest versions of Mid-level-ontology.kif to make it consistent with type information.

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

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

Figure 5. Translation of Second-Order Axioms.

KIF Axioms	FO Formulas
(<=> (instance ?REL SymmetricRelation) (**forall** (?INST$_1$?INST$_2$) (=> (?REL ?INST$_1$?INST$_2$) (?REL ?INST$_2$?INST$_1$)))) (instance relative SymmetricRelation)	Adimen-SUMO: $\forall x_1 \forall x_2$ ($relative(x_1, x_2) \leftrightarrow holds_3(relative', x_1, x_2)$) $\forall x$ ($instance(x, SymmetricRelation) \leftrightarrow$ $\forall y_1 \forall y_2$ ($holds_3(x, y_1, y_2) \rightarrow holds_3(x, y_2, y_1)$)) $instance(relative', SymmetricRelation)$ (Pease & Sutcliffe, 2007): $\forall x$ ($holds_3(instance, x, SymmetricRelation) \leftrightarrow$ $\forall y_1 \forall y_2$ ($holds_3(x, y_1, y_2) \rightarrow holds_3(x, y_2, y_1)$)) $holds_3(instance, relative', SymmetricRelation)$ TPTP-SUMO: $\forall x_1 \forall x_2$ ($relative(x_1, x_2) \rightarrow relative(x_2, x_1)$) $instance(relative', SymmetricRelation)$

like the first one in Figure 5 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, as stated by the second KIF axiom in Figure 5:

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

From a semantic point of view, this translation is equivalent to dealing with structures that comprise an arbitrary set of relations of the domain for interpreting predicate symbols of the alphabet, instead of all the relations that are definable in the domain (like in Henkin semantics) or all possible relations in the domain (like in standard semantics). In some sense, this means to interpret second-order KIF axioms as meta-axioms or axiom schemas. For this purpose, we use the so-called *reflection axioms* (Feferman, 1962), 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 arguments of the predicates $holds_k$.

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

1. For each n-ary predicate symbol r in the alphabet of the ontology, a *reflection axiom* of the form

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

is added, where r' is a new constant symbol associated 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 an argument is replaced with r'.

Note that the use of a new constant symbol (r') that is associated to each predicate symbol (r) in the ontology allows us to express *reflection* using a FO formula while retaining its deductive power.

In Figure 5, we provide the formula that is obtained using our transformation with the characterization of *SymmetricRelation*, where the constant *relative'* is associated to the predicate *relative* (the two other depicted proposals will be commented in the next paragraph). We would like to remark that in the portion of SUMO that we translate, the number of relations is 517, which is the number of reflection axioms that our transformation introduces. Additionally, only 24 axioms require the use of predicates $holds_k$.

In (Pease & Sutcliffe, 2007), the authors also propose to use predicates $holds_k$ in order to translate SO axioms into FO formulas. However, their proposal is completely different from ours since they 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, $holds_k$ are the only predicate symbols in the translated ontology. This fact is illustrated by the example in Figure 5. 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. For example, a simple and efficient axiom selection strategy can be implemented on the basis of a graph of relation dependences (Schlicht & Stuckenschmidt, 2007; Stuckenschmidt & Schlicht, 2009). According to that strategy, only the axioms with relations depending on rela-

Figure 6. Row Variable Elimination

#	KIF Axioms	FO Formulas
1.	(<=> (partition @ROW) (**and** (exhaustiveDecomposition @ROW) (disjointDecomposition @ROW)))	$\forall x_1 \forall x_2 \forall x_3$ ($partition_3(x_1, x_2, x_3) \leftrightarrow$ ($exhaustiveDecomposition_3(x_1, x_2, x_3) \wedge$ $disjointDecomposition_3(x_1, x_2, x_3)$)) $\forall x_1 \forall x_2 \forall x_3 \forall x_4$ ($partition_4(x_1, x_2, x_3, x_4) \leftrightarrow$ ($exhaustiveDecomposition_4(x_1, x_2, x_3, x_4) \wedge$ $disjointDecomposition_4(x_1, x_2, x_3, x_4)$))
2.	(partition Entity Physical Abstract)	$partition_3(Entity, Physical, Abstract)$
3.	(partition Number RealNumber ImaginaryNumber ComplexNumber)	$partition_4(Number, RealNumber, ImaginaryNumber, ComplexNumber)$

tions that occur in the goal to be solved are selected. If only $holds_k$ predicates are used, then the dependence analysis on relations cannot be performed. More sophisticated and time-consuming strategies are required for the proposal of (Pease & Sutcliffe, 2007), as the ones presented in (Hoder & Voronkov, 2011).

On the contrary, in TPTP-SUMO predicates $holds_k$ are not used since axioms with variables as predicates are used as axiom schemes. That is, variable predicates are instantiated to every possible value defined in the ontology, yielding an axiom for each value. The set of TPTP-SUMO formulas that is obtained from the example of *SymmetricRelation* is given in Figure 5. Applying this transformation to the portion of SUMO that is translated in this paper, the number of resulting additional axioms is more than 12000 (24 axioms that use a variable predicate and more than 500 relations). It is important to remark that the total number of axioms in Adimen-SUMO is actually less than 8000. Additionally, the collection of axioms in TPTP-SUMO can be derived from Adimen-SUMO by logical consequence. For instance, it suffices to see that the formula $\forall x_1 \forall x_2$ ($relative(x_1, x_2) \rightarrow relative(x_2, x_1)$) can be deduced from the Adimen-SUMO formulas in Figure 5, since there exists an identical deduction that works for any defined relation r.

Row Variables

In FOL, it is assumed that every predicate and function symbol is used with just one arity and, in addition, predicate and function symbols are disjoint. This is just a syntactic restriction that can be fulfilled when using FO 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. This is the case with the axioms in Figure 6. In Axiom 1, the predicate *partition* is used with the row variable @ROW, where @ROW stands for a tuple of variables with arbitrary size. Additionally, *partition* is used with arities 3 and 4 in the last two axioms.

Thus, the predicates in the last two axioms must be distinct according to FOL syntax. However, the characterization of both predicates (with arities 3 and 4) is given by means of a single axiom (Axiom 1). 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 seven. For example, the above characterization of *partition* that uses a row variable is translated in TPTP-SUMO as follows:

$$\forall x_1 \; (\; partition_1(x_1) \leftrightarrow (\; exhaustiveDecomposition_1(x_1) \wedge disjointDecomposition_1(x_1) \;) \;)$$

$$\forall x_1 \forall x_2 \; (\; partition_2(x_1, x_2) \leftrightarrow (\; exhaustiveDecomposition_2(x_1, x_2) \wedge disjointDecomposition_2(x_1, x_2) \;) \;)$$

$$\ldots$$

$$\forall x_1 \forall x_2 \forall x_3 \forall x_4 \forall x_5 \forall x_6 \forall x_7 \; (\; partition_7(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \leftrightarrow (\; exhaustiveDecomposition_7(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \wedge disjointDecomposition_7(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \;) \;)$$

In the same way, in (Pease & Sutcliffe, 2007) row variables are expanded from 1 to a given number of variables. This translation produces many predicates that are never used in the ontology. A good example is $partition_1$ in the above set of formulas, which indeed defines a non-sense relation: the partition of a class into an empty subset of classes.

In order to avoid the introduction of useless formulas, our approach to translating KIF axioms with row variables (into FO formulas) uses the following iterative process. For each predicate symbol r used in some KIF axiom with row variables, we proceed as follows.[30] First, our translation process

[30]The translation of function symbols follows the same steps.

Figure 7. Definition of Variable Arity Predicates

#	SUMO	Adimen-SUMO
1.	(=> (disjointDecomposition ?CLASS @ROW) (**forall** (?ITEM$_1$?ITEM$_2$) (=> (**and** (inList ?ITEM$_1$ (ListFn @ROW)) (inList ?ITEM$_2$ (ListFn @ROW)) (**not** (equal ?ITEM$_1$?ITEM$_2$))) (disjoint ?ITEM$_1$?ITEM$_2$)))))	(**forall** (?CLASS @ROW) (<=> (disjointDecomposition ?CLASS @ROW) (@**and** (@ROW ?CLASS$_1$ @TAIL) (@**and** (@TAIL ?CLASS$_2$ @_) (disjoint ?CLASS$_1$?CLASS$_2$)))))
2.	(=> (inList ?ITEM ?LIST) (**exists** (?NUMBER) (equal (ListOrderFn ?LIST ?NUMBER) ?ITEM)))	
3.	(=> (**and** (instance ?LIST$_1$ List) (instance ?LIST$_2$ List) (**forall** (?NUMBER) (equal (ListOrderFn ?LIST$_1$?NUMBER) (ListOrderFn ?LIST$_2$?NUMBER)))) (equal ?LIST$_1$?LIST$_2$))	

determines 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 (two row variables in one atom are not allowed) is replaced by an axiom for each possible arity n of r. In the new axioms, the arity of atoms with root predicate r has to be at most n. For this purpose, the row variable is replaced with a tuple of variables of the adequate length, which is determined by n and the arity of the predicate in which the row variable occurs. 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 all of the row variables are eliminated. Note that, after this transformation, it is necessary to rename the predicate symbols that are used with several arities (if any) due to the above-mentioned FOL notational convention.[31]

For example, the result of translating the previous set of axioms containing row variables is given in Figure 6. 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.

In the portion of SUMO translated in this work, 56 axioms contain row variables.

However, a suitable use of row variables also requires additional transformation in SUMO. The problem is related to the characterization of predicates that take row variables as arguments. In Figure 7, we compare the current characterization of the variable arity predicate *disjointDecomposition* in SUMO with our proposal in Adimen-SUMO. In SUMO, *disjointDecomposition* is characterized in Axiom 1 using the predicate *inList*. The axiomatization of *inList* in Axiom 2 essentially characterizes *inList* as an irreflexive and asymmetric binary predicate in terms of the function *ListOrderFn*. In the same way, the partial binary function *ListOrderFn* is poorly axiomatized by Axiom 3. This axiomatization of *ListOrderFn* is not enough for deducing its basic properties. For example, the following assertion cannot be deduced

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

where *ListFn* is a variable-arity function that returns the list with all the subterms as elements. As a consequence, the function *ListOrderFn* and also the predicates *inList*, *disjointDecomposition* and *partition* (and several others) do not produce the expected results. In the proposal of (Pease & Sutcliffe, 2007) and in TPTP-SUMO, this problem remains unsolved. For example, since *partition* is only partially characterized 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*.

We overcome this problem by providing specific operators to deal with row variables: the row operators @**and** and @**or**, which take 3 variables as arguments (2 row variables and a

[31] Additionally, it is also necessary to generate the type information for the new predicates (see Subsection *"Translation of Type Information"*).

Figure 8. Axiomatization of *domain* and *domainSubclass*

Type Information in SUMO

```
(=>
    (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 9. Axiomatization of *temporalPart*

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

single variable). An example of the usage of row operators is given in Figure 7 in the characterization of *disjointDecomposition*. These operators are a kind of iterator that enable us to iteratively process each element in the row variable. The first element (head) in the row variable that occurs in the first argument of a row operator can be addressed using the second argument of the row operator and the remaining elements (@TAIL) are addressed in the third argument, which allows for recursive definitions.[32] Thus, an **@and** (resp. **@or**) formula is translated into a conjunction (resp. disjunction) of formulas, one for each element in the row variable occurring in the first argument. For example, when the row variable @ROW in Figure 7 is instantiated to a tuple of 3 variables, the formula that results from Axiom 1 in Adimen-SUMO is

$$\forall x \forall y_1 \forall y_2 \forall y_3 \; (\; disjointDecompostion4(x, y_1, y_2, y_3) \leftrightarrow \\ (\; disjoint(y_1, y_2) \wedge disjoint(y_1, y_3) \wedge disjoint(y_2, y_3) \;) \;)$$

Alternatively, we can maintain the original characterization of *disjointDecomposition* in SUMO by using row operators **@and** and **@or** to provide a suitable characterization of the predicate *inList* and the function *ListOrderFn*.

Translation of Type Information

In SUMO, there is information that describes the signature of each predicate. That is, the number and argument types 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. As it occurs with every other construct in SUMO, those predicates are also axiomatized in SUMO (see Figure 8). Thus, one possibility to deal with type information in SUMO is to directly translate the axiomatization of *domain* and *domainSubclass* as regular axioms. However, in the current state of SUMO, a direct translation of this type information produces unexpected inconsistencies. Next, we describe this problem in detail. Then, we propose an appropriate translation of type information in SUMO, which is based on the classical translation of many-sorted FO formulas into one-sorted FO formulas. Finally, we describe a structural design problem of SUMO that was discovered after applying our translation of type information.

Unexpected Results from the Direct Translation of Type Information in SUMO. Applying a direct translation to the SUMO axioms, the automatic theorem provers reported many inconsistencies. Surprisingly, these inconsistencies involved semantically correct and well-written axioms. The source of the problem was that a direct translation of SUMO axioms without properly including type information produces too strong formulas.[33] Indeed, in some cases the resulting formulas are even unsatisfiable (possibly in conjunction with other formulas). This is the case with the direct translation of the axioms in Figure 9, that can be proved to be unsatisfiable in the following way. First, the formula

$$instance(temporalPart, ReflexiveRelation)$$

[32] @_ denotes an *anonymous* row variable, following the classical Prolog notation.

[33] A formula ϕ is stronger than ψ if ψ is entailed by ϕ but not vice versa. Likewise, ϕ is weaker than ψ if ϕ is entailed by ψ but not vice versa.

Figure 10. Translation of Type Information

#	KIF Axioms	FO Formulas
	Domain axioms:	
1.	(domain exploits 1 Object)	Axiom 7 without using domain information:
2.	(domain exploits 2 Agent)	$\forall x \forall y$ ($exploits(x, y) \rightarrow \exists z$ ($agent(z, y) \wedge resource(z, x)$))
3.	(domain agent 1 Process)	
4.	(domain agent 2 Agent)	Axiom 7 using domain information:
5.	(domain resource 1 Process)	$\forall x \forall y$ (($instance(x, Object) \wedge instance(y, Agent)$) $\rightarrow$
6.	(domain resource 2 Object)	($exploits(x, y) \rightarrow \exists z$ ($instance(z, Process) \wedge$ $agent(z, y) \wedge$ $resource(z, x)$)))
	Rule axioms:	
7.	(**forall** (?OBJ ?AGENT) (=> (exploits ?OBJ ?AGENT) (**exists** (?PROCESS) (**and** (agent ?PROCESS ?AGENT) (resource ?PROCESS ?OBJ)))))	

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

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

From the above formula and Axiom 7, it follows that

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

which, along with Axiom 5, turns out that

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

Finally, this formula yields an inconsistency for each instance of *Abstract* defined in SUMO, e.g. $instance(YearDuration, Abstract)$. It is easy to see that the formulas obtained from Axioms 1-4 (which provide the type information about *temporalPart* and *time*) have not been used. Thus, the direct translation as regular axioms of type information does not solve the problem.

Next, we describe a translation of type information in SUMO that produces weaker formulas for Axioms 5-10. Indeed, our translation of these axioms allows to deduce the disjointness of the classes *Abstract* and *Physical*, but not the emptiness of *Abstract* as the direct translation of type information as regular axioms allows to infer.

Translation of Type Information into First-Order Formulas. A suitable translation of type information in SUMO is the classical technique that transforms many-sorted FO formulas into equivalent one-sorted FO ones. This technique is described in (Manzano, 1996). Following this proposal, we first distribute universal/existential quantification over conjunction/disjunction in every FO formula ϕ that results from the direct translation of KIF axioms.[34] Then, for each subformula ψ of the form $\forall x(\alpha)$ where α is any formula, we infer the type of x according to the information provided by predicates *domain*, *domainSubclass*, *range* and *rangeSubclass*. Next

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

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

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

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

Similarly, for each subformula ψ of the form $\exists x(\alpha)$ where α is any formula and according to the type information provided by predicates *domain*, *domainSubclass*, *range* and *rangeSubclass*

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

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

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

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

In Figure 10, we illustrate this transformation, comparing the formula that results from our proposal (the second one) with the formula that is obtained by direct translation (the first one), which is weaker. Note that type information restricts x to be an instance of *Object* by Axioms 1 and 6, y to be an instance of *Agent* by Axioms 2 and 4, and z to be an

[34] Note that implication and equivalence can be interpreted as Boolean combinations of disjunction and conjunction with negation.

instance of *Process* by Axioms 3 and 5. In the same way, the translation of Axiom 7 in Figure 9 using the type information in Axioms 1-4 from the same figure 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 in order to prevent the inconsistency reported above regarding the axiomatization of *temporalPart*. In general, by means of the guards, the universal and existential variables in axioms are correctly restricted to their domain, whereas the (separately given) information about domains is not used in deductions as already discussed.

It is worth to note that no type checking is performed before applying the above transformation. In particular, our program does not check if some variable becomes to be instance/subclass of two disjoint classes, which can be considered as a wrongly-typed variable. The effect of a wrongly-typed variable varies depending on whether it is existential or universal. Wrongly-typed existential variables yield an inconsistency (the introduced axiom itself is unsatisfiable) that can be easily detected using FO theorem provers. On the contrary, wrongly-typed universally quantified variables produce tautologies of the form "False implies Formula". Obviously, those types of formulas are not useful for reasoning purposes and, in particular, do not produce any inconsistency.

Regarding other proposals, TPTP-SUMO just performs a direct translation of type information and, in (Pease & Sutcliffe, 2007), the authors describe a transformation that is also intended to solve the problem of making usable the type information in SUMO. However, their translation deals with universally and existentially quantified variables in a uniform way that is semantically incorrect. More specifically, implication is used to connect the guard with the original formula also in the case of existential quantifiers. For example, for each subformula ψ of the form $\exists x(\alpha)$ where α is any formula and the type of x is instance of T, then ψ is replaced with

$$\exists x \; (\; instance(x, T) \rightarrow \alpha \;)$$

which is trivially equivalent to

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

Thus, the existential quantifier can be distributed over disjunction, yielding

$$\exists x \; (\; \neg instance(x, T) \;) \; \vee \; \exists x \; (\; \alpha \;)$$

where the second subformula is ψ. Obviously, this transformation does not properly employ type information.

Still, the axioms obtained using our transformation are not usable for reasoning yet (that is, these axioms do not produce the expected results), because of a self-referentiality problem of SUMO that we explain in the next subsection.

Self-referentiality Problem. 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 illustrate this problem by showing that *Object* cannot be inferred to be a subclass of *Entity*. First, in SUMO *Object* is characterized as a subclass of *Physical*, and *Physical* as a subclass of *Entity*:

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

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

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

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 FO 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, additionally, *PartialOrderingRelation* is a subclass of *TransitiveRelation*, from the above formula we can infer that the predicate *subclass* is an instance of *TransitiveRelation*, provided that *PartialOrderingRelation* is an instance of *SetOrClass*. However, *PartialOrderingRelation* is not characterized in SUMO to be an instance of *SetOrClass*. Thus, the above formula cannot be used to 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 structure the knowledge in SUMO – like *subclass*, *instance*, etc. – are also axiomatized inside SUMO. The self-referential nature of SUMO is illustrated by the following axiom

(instance instance BinaryPredicate)

which is the only characterization of *instance* in the ontology. Obviously, the above axiom is rejected by every FO theorem prover, since it cannot be expressed in FOL. In every other attempt to translate SUMO into FOL (in particular, in the most recent (Pease & Sutcliffe, 2007)), the above axiom is removed and instance becomes undefined, which blocks the deductive process whenever some property of *instance* is required.

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 characterized in terms of them. That is, the remaining "structural" predicates are expressible in terms of $\$instance$ and $\$subclass$. First, the predicate $\$disjoint$ is expressed in terms of $\$instance$ as follows:

$$\forall x \forall y \; (\; \$disjoint(x,y) \leftrightarrow \forall z \; \neg(\; \$instance(z,x) \; \wedge \; \$instance(z,y) \;) \;)$$

Similarly, the predicate $\$exhaustiveDecomposition$ is also definable (see Appendix C). For example, the characterization of $\$exhaustiveDecomposition$ for classes that are decomposed in two subclasses is given by the following:

$$\forall x \forall y_1 \forall y_2 \; (\; \$exhaustiveDecomposition(x,y_1,y_2) \; \leftrightarrow \forall z \; (\; \$instance(z,x) \; \rightarrow \; (\; \$instance(z,y_1) \vee \$instance(z,y_2) \;) \;) \;)$$

Then, the predicates $\$disjointDecomposition$ and $\$partition$ can be expressed in terms of $\$disjoint$ and $\$exhaustiveDecompostion$. For example, for two subclasses, the axiom is:

$$\forall x \forall y_1 \forall y_2 \; (\; \$disjointDecomposition(x,y_1,y_2) \; \leftrightarrow \; \$disjoint(y_1,y_2) \;)$$

$$\forall x \forall y_1 \forall y_2 \; (\; partition(x,y_1,y_2) \leftrightarrow (\; \$exhaustiveDecomposition(x,y_1,y_2) \wedge \$disjointDecomposition(x,y_1,y_2) \;) \;)$$

Hence, the taxonomy of classes in an ontology is FO-axiomatizable in terms of the predicates $\$instance$ and $\$subclass$.

Our proposal consists in translating the whole ontology using a predefined schema. A detailed description of this translation is given in Appendix C. This schema is just a collection of axioms that is commonly used in most ontologies.[35] The schema includes the basic predefined predicates $\$instance$, $\$subclass$, $\$disjoint$ and $\$partition$, which are characterized as usual. Adding the axioms in this schema to SUMO and replacing in the SUMO axioms the predicates *instance*, *subclass*, *disjoint* and *partition* with $\$instance$, $\$subclass$, $\$disjoint$ and $\$partition$ accordingly, all the resulting axioms belong to FOL, also the axiom that characterizes *instance*

($instance instance BinaryPredicate)

where a dollar ('$') is used to mark the predefined predicate. In this way, we obtain Adimen-SUMO, overcoming the self-referentiality problem. Thus, it is now possible to transform a large portion of the SUMO ontology into a set of FO axioms. This solution exhibits one of the main advantages of using FOL to work with ontologies: very simple foundations (or base axiomatizations) are required to define the knowledge described in an ontology. The self-referentiality of SUMO is not mentioned in (Pease & Sutcliffe, 2007). However, the translation in TPTP-SUMO overcomes this problem by means of an axiom that explicitly asserts that every class defined in the ontology is an instance of *SetOrClass*. Obviously, this is an *ad-hoc* solution that, depending on the ontology, could cause serious problems (for example, unsatisfiability if objects and classes are assumed to be disjoint). On the contrary, our solution respects the structural foundations of any taxonomy. Hence, it could be applied to any other ontology, independently of its design or the information that is contained.

Adimen-SUMO

Adimen-SUMO is obtained in a fully automatic way by a translator which, in particular, detects the type of axioms to be processed and the transformations to be applied. Regarding the order in which the transformation steps are applied, firstly our program transforms second-order axioms into first-order axioms. Next, row variables are eliminated, which also yields first-order axioms. Then, our system translates type information, eliminating the self-referential problem of SUMO by means of the predefined predicates. Finally, all the resulting axioms are written in TPTP syntax. As a result of this process, around 88% of the original SUMO (files Merge.kif version 1.78 and Mid-level-ontology.kif version 1.114) axioms have been automatically translated solving all the problems described in this paper.

[35] We use the word schema because different formulas are obtained from our predefined set of axioms depending on the information in the ontology, due to the use of row operators.

Figure 11. Implicit Knowledge in Adimen-SUMO

Goals

a. "Plants do not suffer from headache":
```
(=>
   (attribute ?OBJ Headache)
   (not
      (instance ?OBJ Plant)))
```

b. "The child of an animal cannot be a plant':
```
(=>
   (and
      (instance ?PARENT Animal)
      (parent ?CHILD ?PARENT))
   (not
      (instance ?CHILD Plant)))
```

c. "Plants cannot breathe":
```
(=>
   (instance ?PLANT Plant)
   (not
      (capability Breathing experiencer ?PLANT)))
```

d. "Carnivores do not eat plants":
```
(=>
   (and
      (instance ?CARNIVORE Carnivore)
      (instance ?EATING Eating)
      (agent ?EATING ?CARNIVORE)
      (patient ?EATING ?FOOD))
   (not
      (instance ?FOOD Plant)))
```

By means of this suitable translation of SUMO, we were able to find many inconsistencies like the ones described in Section *"Detecting Inconsistencies"*. We finished correcting SUMO when no more inconsistencies arose from Adimen-SUMO.[36] Thus, Adimen-SUMO could be used by FO theorem provers to establish formal reasoning about the properties and relations of the classes defined by the ontology.

There exist many examples of knowledge that can be inferred using Adimen-SUMO that cannot be obtained using previous FO translations of SUMO. For instance, any property that is a consequence of the axiomatizations of *lists* or the predicate *partition* cannot be inferred from TPTP-SUMO. Actually, TPTP-SUMO explicitly asserts some axioms that should be inferred – hence, it should be implicit knowledge – as a way of repairing this limitation of the translation. An example is the axiom stating the transitivity of the subclass *relation* (see Subsection *"Translation of Type Information: Self-referentiality Problem"*).

In fact, as explained in Section *"Introduction"* with the *brain-plant* example, we plan to exploit Adimen-SUMO in a broad set of knowledge intensive applications. For example, to increase the knowledge contained in WordNet by inferring large volumes of appropriate semantic properties and relations between WordNet synsets. In Figure 11, we provide additional examples of implicit knowledge that can be inferred from Adimen-SUMO (but neither from TPTP-SUMO nor from (Pease & Sutcliffe, 2007) proposals) to illustrate its current reasoning capabilities. Next, we describe these examples in detail:

• The *plant-headache* example (Figure 11.a): "Plants do not suffer from headache" follows from Adimen-SUMO since for suffering from headache it is necessary to have a head:

```
(=>
   (attribute ?E Headache)
   (exists (?H)
      (and
         (instance ?H Head)
         (part ?H ?E)
         (attribute ?H Pain))))
```

• The *parent-child* example (Figure 11.b): "The child of an animal cannot be a plant" since:
```
(=>
   (and
      (parent ?CHILD ?PARENT)
      (subclass ?CLASS Organism)
      (instance ?PARENT ?CLASS))
   (instance ?CHILD ?CLASS))
```

• The *plant-breathing* example (Figure 11.c): "Plants cannot breathe" can be proved because Adimen-SUMO states that the act of breathing requires to have lungs:
```
(=>
   (capability Breathing experiencer ?OBJ)
   (exists (?LUNG)
      (and
         (component ?LUNG ?OBJ)
         (instance ?LUNG Lung))))
```

• The *carnivore-eat-plant* example (Figure 11.d): "Carnivores do not eat plants" follows from Adimen-SUMO since carnivores exclusively eat animals:
```
(=>
   (and
      (instance ?CARNIVORE Carnivore)
      (instance ?EAT Eating)
      (agent ?EAT ?CARNIVORE)
      (patient ?EAT ?PREY))
   (instance ?PREY Animal))
```

The fact that this kind of implicit knowledge is really inferred from Adimen-SUMO confirms that, despite the inherent design problems of SUMO, we have been able to create

[36] That is, current FO theorem provers are not able to find any inconsistency running for a few days, but there is no proof of the existence of a model.

a fully operational version of the ontology by exploiting the current capabilities of FO theorem provers.

Regarding the performance of current FO theorem provers working with Adimen-SUMO and using a standard 64-bit Intel® Core™ i3-2100 CPU @ 3.10GHz desktop machine with 4Gb of ram memory, the *brain-plant* example (see Figure 1) is solved in less than 19 seconds, the *plant-headache* example (Figure 11.a) in less than 193 seconds, the *parent-child* (Figure 11.b) example in less than 57 seconds, the *plant-breathing* (Figure 11.c) example in less than 762 seconds, and the *carnivore-eat-plant* example (Figure 11.d) in less than 194 seconds.

Concluding Remarks

Knowledge representation is a very old field in Artificial Intelligence. There is a tight connection between the formalism for representing knowledge and the inferencing capabilities supported by the formalism.

Although OWL-DL is a very common formal knowledge representation formalism, it is unable to cope with expressive ontologies like SUMO. The development of more expressive ontology languages requires the use of theorem provers able to reason with full first-order logic (FOL) and even its extensions (Horrocks & Voronkov, 2006).

Fortunately, state-of-the-art theorem provers for FOL are highly sophisticated and efficient systems allowing its potential application in Natural Language Processing, Knowledge Engineering, Semantic Web infrastructure, etc.

Moreover, lately a growing interest on FO theorem proving is arising. For instance, the work presented in (Tsarkov, Riazanov, Bechhofer, & Horrocks, 2004) explores the feasibility of using FO theorem provers to compute the inferences that DL (Description Logic) reasoners cannot handle. In another work (Schneider & Sutcliffe, 2011), the authors propose a translation of a fragment of OWL 2 Full (Schneider, 2009) into FOL with the purpose of using automated theorem provers to do reasoning on OWL 2 Full ontologies. They also evaluate the results in an experimental way using different FO automated theorem provers. The results indicate that this approach can be applied in practice for effective OWL reasoning and offers a viable alternative to current Semantic Web reasoners. In (Ramachandran, Reagan, & Goolsbey, 2005), 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 FO theory. The main objective of (Baumgartner & Suchanek, 2006) is to use existing logic programming model generation systems for checking the consistency of FO ontologies. To this end, a translation from FO ontologies into disjunctive logic programs is proposed.

In this paper, we have presented the development of Adimen-SUMO, an operational off-the-shelf first-order ontology. Our main interest is to illustrate in a practical way the use of first-order theorem provers as inference engines for reengineering a large and complex ontology. In particular, we have concentrated our efforts on studying, revising and improving SUMO (Niles & Pease, 2001b) by using Vampire (Riazanov & Voronkov, 2002) and E-Prover (Schulz, 2002). Our main contributions can be summarized as follows. Firstly, we have described in detail the use of FO theorem provers for the fully automatic detection of inconsistencies and also for its correction. Secondly, we have proposed suitable translations into FOL of second order axioms (by means of predicates $holds_k$ and reflection axioms), type information (using guards) and variable-length structures such as row variables (introducing row operators). Finally, we have detected and repaired some important design flaws in SUMO (undefined list predicates and self-referentiality), which has been solved by distinguishing between meta-information and the information defined in the ontology, and by providing the characterization of the structural meta-predicates (instance, subclass, partition, etc.).

It is worth to note that, although we have focused on SUMO in this work, our proposals can be also applied to any other ontology in order to work with FO theorem provers. Indeed, we can easily adapt our translator to other ontologies by simply changing the input language or format. Regarding the concrete features of each ontology, many ontologies contain second-order axioms, type information and variable-length structures. In particular, the ontologies that are written in KIF, and also the ontologies written in languages that have emerged from KIF efforts, such as Common Logic (ISO/IEC International Standard, 2007).[37] With respect to other ontologies, another well-known and large ontology is DOLCE, which is basically a FO ontology, although it also incorporates some axioms from modal logic. There exists a proof of the consistency of the FO part of DOLCE (Kutz & Mossakowski, 2011) that has been made in a modular/structured way using the *Heterogeneous Tool Set (Hets)* (Mossakowski, Maeder, & Lüttich, 2007).[38] However, the modular consistency proof is semi-automatic. Thus, our proposal could be applied to DOLCE in order to confirm that the FO part of DOLCE does not give rise to inconsistencies, but also to translate the modal constructs in DOLCE into FOL and look for inconsistencies in the whole ontology. We guess that $holds_k$ and reflection axioms would be very useful for the last task.

In order to maximally exploit existing semantic resources, such as those already integrated into the Multilingual Central Repository[39] (MCR) (Gonzalez-Agirre et al., 2012a) or connected to Linked Data (Bizer, Heath, & Berners-Lee, 2009),

[37] http://www.iso-commonlogic.org

[38] http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/CoFI/hets/index_e.htm

[39] http://adimen.si.ehu.es/web/MCR

new capabilities for reasoning and logical inference should be applied. These capabilities are essential for verifying meta-properties, like consistency, and for inferring new implied properties and relations from the combination of available semantic resources and formal ontologies. That is, beyond the literal meaning expressed in formal ontologies, an intelligent system needs to know what the implications of that meaning are.

In the near future, we plan to exploit Adimen-SUMO and its complete mapping to WordNet for reasoning about the implicit and explicit knowledge contained in the Multilingual Central Repository (MCR), in a similar way as with the EuroWordNet Top Ontology (Álvez et al., 2008). Applying efficient theorem provers, Adimen-SUMO can be directly used for automatically inferring new semantic properties and relations between WordNet concepts. Furthermore, since the MCR integrates different wordnets in different languages via the Inter-Lingual-Index (ILI) (Vossen, 1998), Adimen-SUMO inferences can also be of utility to language resources other than English.

Our final goal is to provide formal underpinnings and advanced reasoning capabilities to existing semantic resources and advanced Natural Language Processing, Knowledge Engineering and Semantic Web tasks.

Acknowledgement

We thank the anonymous reviewers for their valuable comments and suggestions. We are also grateful to Geoff Sutcliffe and Adam Pease for revising preliminary versions of this paper. This work has been partially supported by the KYOTO project[40] (ICT-211423), the Spanish projects KNOW-2[41] (TIN2009-14715-C04-01) and FORMALISM (TIN2007-66523), and the Basque Project LoRea (GIU07/35).

References

Abadi, M. (1989). The power of temporal proofs. *Theoretical Computer Science*, *65*(1), 35–83.

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, *23*, 123–154.

Álvez, J., Atserias, J., Carrera, J., Climent, S., Laparra, E., Oliver, A., et al. (2008). Complete and consistent annotation of WordNet using the Top Concept Ontology. In N. Calzolari et al. (Eds.), *Proceedings of the 6th International Language Resources and Evaluation (LREC 2008)* (pp. 1529–1534). European Language Resources Association (ELRA).

Andréka, H., Németi, I., & van Benthem, J. (1998). Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, *27*, 217–274.

Atserias, J., Rigau, G., & Villarejo, L. (2004). Spanish WordNet 1.6: Porting the Spanish WordNet across Princeton versions. In N. Calzolari et al. (Eds.), *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)* (pp. 161–164). European Language Resources Association (ELRA).

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., & Ives, Z. (2007). DBpedia: A nucleus for a web of open data. In K. Aberer et al. (Eds.), *The Semantic Web - Proceedings of the 6th International Semantic Web Conference (ISWC 2007) and the 2nd Asian Semantic Web Conference (ASWC 2007)* (Vol. 4825, pp. 722–735). Springer Berlin Heidelberg.

Baumgartner, P., Fuchs, A., & Tinelli, C. (2006). Implementing the Model Evolution Calculus. *International Journal on Artificial Intelligence Tools*, *15*(1), 21–52.

Baumgartner, P., & Suchanek, F. (2006). Automated reasoning support for first-order ontologies. In J. Alferes, J. Bailey, W. May, & U. Schwertel (Eds.), *Principles and practice of semantic web reasoning* (Vol. 4187, pp. 18–32). Springer Berlin Heidelberg.

Bentivogli, L., Forner, P., Magnini, B., & Pianta, E. (2004). Revising the WordNet domains hierarchy: semantics, coverage and balancing. In G. Sérasset, S. Armstrong, C. Boitet, A. Popescu-Belis, & D. Tufis (Eds.), *Proceedings of the Workshop on Multilingual Linguistic Ressources (MLR 2004)* (pp. 94–101). Association for Computational Linguistics.

Benzmüller, C., & Pease, A. (2012). Higher-order aspects and context in SUMO. *Web Semantics: Science, Services and Agents on the World Wide Web*, *12*.

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, *5*(3), 1–22.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., et al. (2009). DBpedia - a crystallization point for the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, *7*(3), 154–165.

Borgo, S., Guarino, N., & Masolo, C. (1996). A pointless theory of space based on strong connection and congruence. In L. C. Aiello, J. Doyle, & S. C. Shapiro (Eds.), *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR'96)* (pp. 220–229). Morgan Kaufmann.

Borgo, S., Guarino, N., & Masolo, C. (1997). An ontological theory of physical objects. In L. Ironi (Ed.), *Proceedings of 11th International Workshop on Qualitative Reasoning (QR'97)* (pp. 223–231).

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

Claessen, K., & Sörensson, N. (2003). New techniques that improve MACE-style model finding. In *Proceedings of Workshop on Model Computation (MODEL 2003).*

Cohen, P. R., & Levesque, H. J. (1990). Persistence, intention, and commitment. In P. R. Cohen, J. Morgan, & M. E. Pollack (Eds.), *Intentions in Communication* (pp. 33–69). Cambridge, MA: MIT Press.

d'Aquin, M., & Noy, N. F. (2012). Where to publish and find ontologies? A survey of ontology libraries. *Web Semantics: Science, Services and Agents on the World Wide Web*, *11*, 96–111.

Feferman, S. (1962). Transfinite recursive progressions of ax-

[40] `http://www.kyoto-project.eu`

[41] `http://ixa.si.ehu.es/know2`

iomatic theories. *The Journal of Symbolic Logic*, *27*(3), 259–316.

Fellbaum, C. (Ed.). (1998). *WordNet: An electronic lexical database*. MIT Press.

Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., & Schneider, L. (2002). Sweetening ontologies with DOLCE. In A. Gómez-Pérez & V. R. Benjamins (Eds.), *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002)* (Vol. 2473, pp. 166–181). Springer Berlin Heidelberg.

Genesereth, M. R., Fikes, R. E., Brobow, D., Brachman, R., Gruber, T., Hayes, P., et al. (1992). *Knowledge Interchange Format version 3.0 reference manual* (Tech. Rep. No. Logic-92-1). Stanford University, Computer Science Department, Logic Group.

Gonzalez-Agirre, A., Laparra, E., & Rigau, G. (2012a). Multilingual Central Repository version 3.0. In N. Calzolari et al. (Eds.), *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)* (pp. 2525–2529). European Language Resources Association (ELRA).

Gonzalez-Agirre, A., Laparra, E., & Rigau, G. (2012b). Multilingual Central Repository version 3.0: upgrading a very large lexical knowledge base. In C. Fellbaum & P. Vossen (Eds.), *Proceedings of the 6th Global WordNet Conference (GWC 2012)*.

Gruber, T. (2009). Ontology. In L. Liu & M. T. Özsu (Eds.), *Encyclopedia of database systems* (pp. 1963–1965). Springer US.

Harabagiu, S. M., & Moldovan, D. I. (1998). Knowledge processing on an extended WordNet. In *(Fellbaum, 1998)* (pp. 379–405). MIT Press.

Hayes, P., & Menzel, C. (2001). A semantics for the Knowledge Interchange Format. In *Proceedings of IJCAI 2001 Workshop on the IEEE Standard Upper Ontology*.

Henkin, L. (1950). Completeness in the theory of types. *The Journal of Symbolic Logic*, *15*(2), 81–91.

Hoder, K., & Voronkov, A. (2011). Sine qua non for large theory reasoning. In N. Bjørner & V. Sofronie-Stokkermans (Eds.), *Proceedings of the 23rd International Conference on Automated Deduction (CADE-23)* (Vol. 6803, pp. 299–314). Springer Berlin Heidelberg.

Hoffart, J., Suchanek, F. M., Berberich, K., Lewis-Kelham, E., Melo, G. de, & Weikum, G. (2011). YAGO2: Exploring and querying world knowledge in time, space, context, and many languages. In S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, & R. Kumar (Eds.), *Proceedings of the 20th International Conference Companion on World Wide Web (WWW 2011)* (pp. 229–232). New York, NY, USA: ACM.

Hoffart, J., Suchanek, F. M., Berberich, K., & Weikum, G. (2013). YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, *194*, 28–61.

Horrocks, I., & Patel-Schneider, P. (2004). Reducing OWL entailment to description logic satisfiability. *Web Semantics: Science, Services and Agents on the World Wide Web*, *1*(4), 345–357.

Horrocks, I., & Voronkov, A. (2006). 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, pp. 201–218). Springer Berlin Heidelberg.

ISO/IEC International Standard. (2007). *Information technology – Common Logic (CL): A framework for a family of logic-based languages* (No. ISO/IEC 24707:2007(E)).

Jain, P., Hitzler, P., Sheth, A., Verma, K., & Yeh, P. (2010). Ontology alignment for Linked Open Data. In P. Patel-Schneider et al. (Eds.), *The Semantic Web - Proceedings of the 9th International Semantic Web Conference (ISWC 2010)* (Vol. 6496, pp. 402–417). Springer Berlin Heidelberg.

Jain, P., Hitzler, P., Yeh, P. Z., Verma, K., & Sheth, A. P. (2010). Linked Data is merely more data. In D. Brickley, V. K. Chaudhri, H. Halpin, & D. McGuinness (Eds.), *Proceedings of the AAAI Spring Symposium: Linked Data Meets Artificial Intelligence* (pp. 82–86). Menlo Park, California: AAAI Press.

Keisler, H. J. (1971). *Model theory for infinitary logic: Logic with countable conjunctions and finite quantifiers* (Vol. 62 Studies in Logic and the Foundations of Mathematics). Amsterdam: North-Holland Pub. Co.

Kobilarov, G., Bizer, C., Auer, S., & Lehmann, J. (2009). DBpedia - A Linked Data hub and data source for web applications and enterprises. In R. Daruwala & C. Yu (Eds.), *Proceedings of developers track of the 18th International World Wide Web Conference (WWW 2009)*.

Korovin, K. (2008). 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, pp. 292–298). Springer Berlin Heidelberg.

Kutz, O., & Mossakowski, T. (2011). A modular consistency proof for DOLCE. In W. Burgard & D. Roth (Eds.), *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI 2011)*. Menlo Park, California: AAAI Press.

Magnini, B., & Cavaglià, G. (2000). Integrating subject field codes into WordNet. In A. Zampolli et al. (Eds.), *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)* (pp. 1413–1418). European Language Resources Association (ELRA).

Manzano, M. (1996). *Extensions of first-order logic* (No. 19 Cambridge Tracts in Theoretical Computer Science). Cambridge, UK: Cambridge University Press.

Matuszek, C., Cabral, J., Witbrock, M. J., & DeOliveira, J. (2006). An introduction to the syntax and content of Cyc. In C. Baral (Ed.), *Proceedings of the AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering* (pp. 44–49). Menlo Park, California: AAAI Press.

McCune, W. (2001). *Mace 2.0 reference manual and guide* (Tech. Rep. No. ANL/MCS-TM-249). Mathematics and Computer Science Division, Argonne National Laboratory.

McCune, W., & Wos, L. (1997). Otter - the CADE-13 competition incarnations. *Journal of Automated Reasoning*, *18*, 211–220.

Menzel, C., & Hayes, P. (2003). SCL: A logic standard for semantic integration. In A. Doan, A. Halevey, & N. Noy (Eds.), *Proceedings of the Semantic Integration Workshop* (Vol. 82). CEUR-WS.org.

Mossakowski, T., Maeder, C., & Lüttich, K. (2007). The heterogeneous tool set, Hets. In O. Grumberg & M. Huth (Eds.), *Proceedings of the 13th International Conference in Tools and Al-*

gorithms for the Construction and Analysis of Systems (TACAS 2007) (Vol. 4424, pp. 519–522). Springer Berlin Heidelberg.

Motik, B., Shearer, R., & Horrocks, I. (2009). Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, *36*(1), 165–228.

Niles, I., & Pease, A. (2001a). Origins of the IEEE Standard Upper Ontology. In *Working notes of the Workshop on the IEEE Standard Upper Ontology (IJCAI 2001)* (pp. 37–42).

Niles, I., & Pease, A. (2001b). Towards a standard upper ontology. In N. Guarino, C. Welty, & B. Smith (Eds.), *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS 2001)* (pp. 2–9). New York, NY, USA: ACM.

Niles, I., & Pease, A. (2003). Linking lexicons and ontologies: Mapping WordNet to the Suggested Upper Merged Ontology. In H. R. Arabnia (Ed.), *Proceedings of the IEEE International Conference on Information and Knowledge Engineering (IKE 2003)* (Vol. 2, pp. 412–416). CSREA Press.

Noy, N. F., & McGuinness, D. L. (2001). *Ontology development 101: A guide to creating your first ontology* (Tech. Rep. No. KSL-01-05 and SMI-2001-0880). Stanford Knowledge Systems Laboratory and Stanford Medical Informatics.

Oberle, D., Ankolekar, A., Hitzler, P., Cimiano, P., Sintek, M., Kiesel, M., et al. (2007). DOLCE ergo SUMO: On foundational and domain models in the SmartWeb integrated ontology (SWIntO). *Web Semantics: Science, Services and Agents on the World Wide Web*, *5*(3), 156–174.

Pease, A. (2009). *Standard Upper Ontology Knowledge Interchange Format.* (Retrieved June 18, 2009, from `http://sigmakee.cvs.sourceforge.net/*checkout*/sigmakee/sigma/suo-kif.pdf`)

Pease, A., & Fellbaum, C. (2010). Formal ontology as interlingua: the SUMO and WordNet linking project and global WordNet. In C. Huang, N. Calzolari, A. Gangemi, A. Lenci, A. Oltramari, & L. Prevot (Eds.), *Ontology and the lexicon - A natural language processing perspective* (pp. 25–35). Cambridge University Press.

Pease, A., & Sutcliffe, G. (2007). First-order reasoning on a large ontology. In G. Sutcliffe, J. Urban, & S. Schulz (Eds.), *Proceedings of the Workshop on Empirically Successful Automated Reasoning in Large Theories (CADE-21)* (Vol. 257). CEUR-WS.org.

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

Ramachandran, D., Reagan, R. P., & Goolsbey, K. (2005). First-orderized ResearchCyc: Expressivity and efficiency in a common-sense ontology. In P. Shvaiko, J. Euzenat, A. Leger, D. L. McGuinness, & H. Wache (Eds.), *Papers from the AAAI 2005 Workshop on Contexts and Ontologies: Theory, Practice and Applications* (pp. 33–40). Menlo Park, California: AAAI Press.

Reed, S. L., & Lenat, D. B. (2002). Mapping ontologies into Cyc. In A. Pease (Ed.), *Papers from AAAI 2002 Workshop on Ontologies For The Semantic Web* (pp. 1–6). Menlo Park, California: AAAI Press.

Riazanov, A., & Voronkov, A. (2002). The design and implementation of Vampire. *AI Communications*, *15*(2-3), 91–110.

Schlicht, A., & Stuckenschmidt, H. (2007). Criteria-based partitioning of large ontologies. In D. H. Sleeman & K. Barker (Eds.), *Proceedings of the 4th International Conference on Knowledge Capture (K-CAP 2007)* (pp. 171–172). New York, NY, USA: ACM.

Schneider, M. (Ed.). (2009). *OWL 2 Web Ontology Language RDF-based semantics.* (Retrieved October 27, 2009, from `http://www.w3.org/TR/owl2-rdf-based-semantics/`)

Schneider, M., & Sutcliffe, G. (2011). Reasoning in the OWL2 full ontology language using first-order automated theorem proving. In N. Bjørner & V. Sofronie-Stokkermans (Eds.), *Proceedings of the 23rd International Conference on Automated Deduction (CADE-23)* (Vol. 6803, pp. 461–475). Springer Berlin Heidelberg.

Schulz, S. (2002). E - A brainiac theorem prover. *AI Communications*, *15*(2-3), 111–126.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, *5*(2), 51–53.

Sowa, J. F. (2000). *Knowledge representation: Logical, philosophical, and computational foundations*. Pacific Grove, CA: Brooks Cole Publishing Co.

Staab, S., & Studer, R. (Eds.). (2009). *Handbook on ontologies (2nd edition)*. Springer.

Streeter, M. J., Golovin, D., & Smith, S. F. (2007). Combining multiple heuristics online. In R. C. Holte & A. Howe (Eds.), *Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (Vol. 2, pp. 1197–1203). Menlo Park, California: AAAI Press.

Stuckenschmidt, H., & Schlicht, A. (2009). Structure-based partitioning of large ontologies. In H. Stuckenschmidt, C. Parent, & S. Spaccapietra (Eds.), *Modular ontologies: Concepts, theories and techniques for knowledge modularization* (Vol. 5445, pp. 187–210). Springer Berlin Heidelberg.

Suchanek, F. M., Kasneci, G., & Weikum, G. (2007). Yago: A core of semantic knowledge. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, & P. J. Shenoy (Eds.), *Proceedings of the 16th International World Wide Web conference (WWW 2007)* (pp. 697–706). New York, NY, USA: ACM.

Sutcliffe, G. (2009). The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, *43*(4), 337–362.

Sutcliffe, G., & Benzmüller, C. (2010). Automated reasoning in higher-order logic using the TPTP THF infrastructure. *Journal of Formalized Reasoning*, *3*(1), 1–27.

Sutcliffe, G., & Suttner, C. (2006). The state of CASC. *AI Communications*, *19*(1), 35–48.

Tsarkov, D., & Horrocks, I. (2006). FaCT++ description logic reasoner: system description. In U. Furbach & N. Shankar (Eds.), *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR 2006)* (Vol. 4130, pp. 292–297). Springer Berlin Heidelberg.

Tsarkov, D., Riazanov, A., Bechhofer, S., & Horrocks, I. (2004). Using Vampire to reason with OWL. In S. A. McIlraith, D. Plexousakis, & F. Harmelen (Eds.), *Proceedings of the 3rd International Semantic Web Conference (ISWC 2004)* (Vol. 3298, pp. 471–485). Springer Berlin Heidelberg.

Verdezoto, N., & Vieu, L. (2011). Towards semi-automatic methods for improving WordNet. In J. Bos & S. Pulman (Eds.), *Proceedings of the 9th International Conference on Computational*

Semantics (IWCS 2011) (pp. 275–284). Portland, Oregon, USA: Association for Computational Linguistics.

Vossen, P. (1998). *EuroWordNet: A multilingual database with lexical semantic networks*. Norwell, MA, USA: Kluwer Academic Publishers.

Appendix A
Analysing Traces of FO Theorem Provers

In this appendix, we show a trace obtained by E-Prover for the example of Section *"Detecting Inconsistencies"*. 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 interested reader can easily reproduce the trace using 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
```

This command uses the default configuration of E-Prover (that is, the system can use any axiom/atom selection strategy). In Figure A1, 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. In addition, 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 formulas in steps 18, 28, 35 and 39 are disjuncts obtained from the transformation of the formulas in steps 1-4, where the disjuncts that will not be used in the trace have 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. The formulas in steps 40-46 are directly obtained from steps 5-10. Then, E-Prover starts to obtain new formulas by resolution. On one hand, the formula `++instance(outdoors,nonNullSet)` (step 71) follows from `++instance(X2,nonNullSet)` (in step 18) and `++disjoint(indoors,outdoors)` (in step 44). On the other hand, the formula `-instance(outdoors,nonNullSet)` (in step 179) is obtained after ten resolution steps. To sum up, from the formulas in steps 39, 41 and 42 it follows that everything is either `physical` or not `region` (in step 154) after two resolution steps. Hence, we have that `++instance(outdoors,physical)` (in step 164) since `++instance(outdoors,region)` (from step 45). Additionally, from the formulas 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 formulas 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, we easily have that `-instance(outdoors,nonNullSet)` (in step 179). Hence, the formulas in steps 71 and 179 yield the inconsistency `-true` (`++false`) in step 180.

Appendix B
Using Adimen-SUMO for Automated Reasoning

Here we show the trace that is obtained by E-Prover for the example described in Section *"Introduction"* about plants and brains. Figure B1 shows 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 formulas (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. Furthermore, the disjuncts in steps 48, 53 and 71 are obtained by transforming the formulas in 6, 7 and 9, where the disjuncts that are not going to be used in the proof have been discarded. The remaining formulas of the initial transformation are the disjuncts in steps 40, 74, 75, 81 and 84, which are directly obtained from the formulas in steps 5, 10, 11, 17 and 20, respectively.

The inconsistency comes from both `-instance(esk1_0,animal)` (in step 387) and `++instance(esk1_0,animal)` (in step 442). On one hand, `-instance(esk1_0,animal)` results from a 4 step resolution sequence that starts with the formulas in steps 53 and 84 to finally yield `++disDecomp4(organism,animal,plant, microorganism)` (in step 215). Then, using the formula in step 71, it obtains `++disjoint(animal, plant)` (in step 340). Next, using the formula in step 48, it follows that everything is not an instance of either `plant` or `animal`. Hence, since `++instance(esk1_0,plant)` (in step 26), the

Figure A1. An Inconsistency Discovered by E-Prover

```
  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'
```

formula -instance(esk1_0,animal) is inferred at step 387. On the other hand, ++instance(esk1_0,animal) results from an 8 step resolution sequence. First, from the formulas in steps 40, 75 and 81, it follows that everything is an instance of animalAS or not an instance of brain (in step 157), and also that everything is 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), it respectively obtains the formulas ++instance(esk2_0,animalAS) (in step 239) and ++instance(esk1_0,organism) (in step 249). Using these last two formulas and also the formulas ++part(esk2_0,esk1_0), ++instance(esk1_0,object) and ++instance(esk2_0,object) (in steps 25, 28 and 29 respectively), it infers ++instance(esk1_0,animal) (in step 442) from the disjunct in step 74 (after resolution steps 183, 436, 439 and 440). Hence, a contradiction is achieved at step 443, that proves the original goal.

Appendix C
Translating SUMO into FO Formulas: A Detailed Example

In this appendix, we provide a detailed example of the translation of SUMO into FO formulas. Here, we especially focus on some details that have not been presented in the body of the paper. We choose TPTP syntax to write FO formulas, since most of current FO theorem provers accept it. Here, we just use the existential (?) and universal (!) quantifiers and the classical connectives of negation (~), conjunction (&), disjunction (|), implication (=>) and equivalence (<=>). A whole description of TPTP syntax is available at http://www.tptp.org.

Figure B1. Do Plants Have Brain?

```
  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", merge178B1)
 11 :  :  subclass(brain,animalAS) : initial("brain.eprover.tstp", milo1114B1)
 17 :  :  subclass(plant,organism) :  initial("brain.eprover.tstp", merge178B7)
 20 :  :  partition4(organism,animal,plant,microorganism) :  initial("brain.eprover.tstp", merge178B10)
 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 C1. Definition of *Meta-predicates* in Adimen-SUMO

#	Axioms
1.1	($domain $instance 1 $object)
1.2	($domain $instance 2 $class)
1.3	($domain $subclass 1 $class)
1.4	($domain $subclass 2 $class)
1.5	(**forall** (?X) ($subclass ?X ?X))
1.6	(**forall** (?X?Y?Z) (=> ($subclass ?X ?Y) ($subclass ?Y ?Z)) ($subclass ?X ?Z))
1.7	(**forall** (?X?Y) (=> ($subclass ?X ?Y) ($subclass ?Y ?X)) (equal ?X ?Y))
1.8	(**forall** (?X?Y?Z) (=> ($instance ?X ?Y) ($subclass ?Y ?X)) ($instance ?X ?Z))

As explained in Section *"Translating SUMO into First-Order Logic"*, we translate first-order axioms, second-order axioms, axioms containing row variables and type information. Additionally, in Adimen-SUMO we introduce new axioms that provide the axiomatization of meta-predicates ($*instance* and $*subclass*), as shown in Figure C1, and predefined predicates ($*disjoint*, $*partition*, $*exhaustiveDecomposition* and $*disjointDecomposition*), as shown in Figure C2. Note that the axiomatizations of both meta-predicates and predefined predicates include type information, which is provided using the predicate $*domain* and also the predefined constants $*object* and $*class*, which respectively correspond to the meta-level concepts of object and class.

In order to illustrate the translation of SUMO into FO formulas, together with the axioms in Figures C1 and C2, we also consider the set of axioms in Figure C3. In this example, $*holds*3 is used to write the reflexive property of binary relations in FOL (axiom 6). First, since the arities of the row lists in axioms 3.1, 3.2 and 3.33 are 5, 3 and 2, respectively, axioms 2.3, 2.6 and 2.9 have to be translated according to each arity. Then, axioms 3.1-3.3 are directly translated into:

Figure C2. Definition of Predefined Predicates in Adimen-SUMO

#	Axioms
2.1	($domain $partition 1 $class)
2.2	($domain $partition 2 @$class)
2.3	(**forall** (?CLASS @ROW) (<=> ($partition ?CLASS @ROW) (=> ($exhaustiveDecomposition ?CLASS @ROW) ($disjointDecomposition ?CLASS ?ROW))))
2.4	($domain $exhaustiveDecomposition 1 $class)
2.5	($domain $exhaustiveDecomposition 2 @$class)
2.6	(**forall** (?CLASS @ROW) (<=> ($exhaustiveDecomposition ?CLASS @ROW) (**forall** (?X) (=> ($instance ?X ?CLASS) (@**or** (@ROW, ?HEAD, @_) ($instance ?X ?HEAD))))))
2.7	($domain $disjointDecomposition 1 $class)
2.8	($domain $disjointDecomposition 2 @$class)
2.9	(**forall** (?CLASS @ROW) (@**and** (@ROW, ?CLASS1, @TAIL) (@**and** (@TAIL, ?CLASS2, @_) ($disjoint ?CLASS1 ?CLASS2)))))
2.10	($domain $disjoint 1 $class)
2.11	($domain $disjoint 2 $class)
2.12	(**forall** (?CLASS1 ?CLASS2) (<=> ($disjoint ?CLASS1 ?CLASS2) (**forall** (?INST) (**not** (**and** ($instance ?INST ?CLASS1) ($instance ?INST ?CLASS2))))))

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

$partition4(Relation,Predicate,Function,List)

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

Figure C3. An Example Extracted from Adimen-SUMO

#	Axioms
3.1	($disjointDecomposition Relation @(BinaryRelation, TernaryRelation, QuaternaryRelation, QuintaryRelation, VariableArityRelation))
3.2	($partition Relation 1 @(Predicate, Function, List))
3.3	($partition Relation 1 @(TotalValuedRelation, PartialValuedRelation))
3.4	($subclass BinaryRelation 1 Relation)
3.5	($subclass ReflexiveRelation 1 BinaryRelation)
3.6	(<=> ($instance ?REL ReflexiveRelation) (**forall** (?INST) ($holds3 ?REL ?INST ?INST)))
3.7	($domain connected 1 Object)
3.8	($domain connected 2 Object)
3.9	($instance connected BinaryPredicate)
3.10	($instance connected ReflexiveRelation)
3.11	($subrelation meetsSpatially connected)
3.12	($subrelation overlapsSpatially connected)
3.13	(=> (connected ?OBJ1 ?OBJ2) (**or** (meetsSpatially ?OBJ1 ?OBJ2) (overlapsSpatially ?OBJ1 ?OBJ2)))

Then, the translation of axioms 3.4-3.6 is also direct:

```
$subclass(BinaryRelation,Relation)

$subclass(ReflexiveRelation,BinaryRelation)

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

Next, we translate axioms 3.9, 3.10 and 3.13. Note that, since the predicates *meetsSpatially* and *overlapsSpatially* are subrelations of *connected*, type information for *meetsSpatially* and *overlapsSpatially* is inherited from axioms 3.7-3.8. Further, we use *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*, *meetsSpatially* and *overlapsSpatially* with $*holds*3 via the constants *constConnected*, *constMeetsSpatially*, *constOverlapsSpatially*:

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

(![X,Y]: meetsSpatially(X,Y) <=>
             $holds3(constMeetsSpatially,X,Y))

(![X,Y]: overlapsSpatially(X,Y) <=>
             $holds3(constOverlapsSpatially,X,Y))
```

Appendix D
Resources

In this appendix, we briefly describe the files contained in the Adimen-SUMO package[42] that have been used along the work reported in this paper. The interested reader may try the examples described in this work, and many others, using a FO theorem prover such as e.g. E-Prover or Vampire.

The translator implemented in Prolog is included in "E-KIFtoFOF" folder. The main file is **E-KIFtoFOF.pl**.

The files that contain the ontology in KIF format are **predefinitions.kif**, **merge1.78.kif** and **milo1.114.kif**. The file **predefinitions.kif** contains the definitions of the basic predicates that are used in the rest of the ontology, as described at the end of Section *"Translating SUMO into First-Order Logic"*. The syntax of this file also includes some non KIF features, such as row operators (see Subsection *"Row Variables"* and 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 files **merge1.78.kif** and **milo1.114.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 designed to be used with E-Prover.[43] This file has been tested using several FO

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

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

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 Adimen-SUMO.

An example of inconsistency that has been found by a FO theorem prover in a preliminary version of **adimen.sumo.eprover.tstp** is described in Section *"Detecting Inconsistencies"* (see also Appendix A). The axioms that are necessary to reproduce the inconsistency have been collected in the file **disjoint.eprover.tstp**. The whole refutation provided by E-Prover for this inconsistency can be consulted in **output.disjoint.eprover.txt**.

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