A New Proposal for Using First-Order Theorem Provers to Reason with OWL DL Ontologies [1]

M. Alecha[2], J. Álvez[2,3], M. Hermo[2] and E. Laparra[2]

Dpto. de Lenguajes y Sistemas Informáticos
Universidad del País Vasco

Dpto. de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Abstract

Existing OWL DL reasoners have been carefully designed to reason with DL ontologies in an efficient way at the expense of lack of expressiveness. In order to overcome this limitation in expressiveness, there have been a few attempts to use first-order logic (FOL) theorem provers, which are known to be less efficient than DL reasoners, to work with DL ontologies. However, these approaches did not still allow full FOL capabilities in queries. In this paper, we introduce a new approach (currently under development) to translate OWL DL ontologies into FOL. The translation has been tested using a simple ontology about animals and some FOL theorem provers. On the basis of these tests, we show that our proposal achieves a good trade-off between expressive and simple queries. On one hand, our system is capable of handling any FOL query that can not be processed by DL reasoners. On the other hand, simple queries are solved in reasonable time by FOL theorem provers in comparison with *ad hoc* DL reasoners.

Keywords: OWL DL, first-order logic, theorem proving, ontologies.

1 Introduction

The OWL language [3] has been developed to publish and share knowledge on the Web using ontologies. More specifically, there are three different sublanguages of OWL: OWL Lite, OWL DL and OWL Full. Each of these sublanguages has a different level of expressiveness, OWL Lite being the least expressive one. The OWL DL sublanguage corresponds to a decidable fragment of FOL: the $\mathcal{SHOIN}\mathcal{D}_n^-$ Description Logic (DL) [1,3,5].

Currently, there are some efficient OWL DL reasoners, such as FaCT++ [14] or Pellet [12], which are sound and complete. Besides, there exist some very powerful utilities for creating, maintaining and accessing to DL ontologies, such as Protégé [7]. All these tools provide the use of DL ontologies for representing knowledge. However, the lack of expressiveness of OWL DL drastically reduces its application for dealing with complex information and it is commonly accepted that DL ontologies are only suitable for representing very simple relations and properties. For example, the transitivity property of relations

[1] This work has been partially supported by the Spanish projects TIN2007-66523, LoRea (GIU07/35), Promesas-CAM (S-0505/TIC/0407) and STAMP (TIN2008-06622-C03-01/TIN).

[2] Email: {mikel.alecha, javier.alvez, montserrat.hermo, egoitz.laparra}@ehu.es

[3] Email: javieralvez@fdi.ucm.es

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

can not be expressed using the OWL DL language, although transitivity is used in OWL DL as a built-in property (inheritable relations). In the same way, one could try to simulate any property that can not be expressed in OWL DL. However, this solution is not suitable because it requires to adapt DL reasoners to each simulated property using *ad hoc* techniques.

In the literature, we can find some attempts to use FOL theorem provers to reason with DL ontologies [2,15,16]. Since OWL DL is a fragment of FOL, these approaches propose different translations of DL axioms into a FOL language. Then, we can use any general purpose FOL theorem prover to reason with the translated ontology. Obviously, DL reasoners outperform FOL theorem provers at solving OWL DL queries, specially when dealing with large ontologies. However, these past studies have shown that it could be a good idea to generate hybrid systems for reasoning with DL ontologies. Following such a proposal, DL reasoners would handle most of the tasks —consistency checking, ontology classification, OWL DL queries, etc.—, whereas FOL theorem provers would only be used when strictly necessary. That is, to solve non-OWL DL queries. In this task, the performance of FOL theorem provers highly depends on the translation from DL into FOL axioms. Indeed, in order to achieve better performance, the syntactic form of queries is restricted in the proposals in [2,16] (non-OWL DL queries are out of the scope of [15]). Thus, the lack of expressiveness in queries still remains when reasoning with DL ontologies.

In this work, we present a different proposal for the translation of OWL DL ontologies into FOL. The main purpose of our translation is to be able to use FOL theorem provers for reasoning with DL ontologies in an efficient way without restricting queries to the OWL DL sublanguage. Most successful FOL theorem provers, such as Vampire [10] or E Prover [11], work by refutation. That is, given any theory and any goal, theorem provers try to decide whether the goal is a logical consequence of the theory by proving that the conjunction of the theory and the negation of the goal is unsatisfiable. The main drawback is that these systems are able to answer whenever the goal follows from the theory (the answer is positive), but theorem provers usually loop (give no answer) when there is no refutation. In order to overcome this limitation, our proposal consists in translating any OWL DL ontology into a decidable and complete FOL theory. In this way, given any goal, we ask to the theorem prover whether the goal or the negated goal is a logical consequence of the theory. That is, we run two queries in parallel. Since the theory is decidable and complete, theorem provers are supposed to find a refutation for some of the queries, which allows us to solve any goal.

We have tested our translation with an small ontology about animals that has been extracted from the SUMO ontology [8]. For the tests, we have used Vampire [10], which is one of the most successful and efficient FOL theorem provers of the CASC competition [9,13]. Our results prove that the proposed system is suitable for reasoning with DL ontologies.

The paper is organized as follows: in the next section, we briefly describe the OWL DL language and provide some notation; besides, in Subsection 2.1, we introduce the ontology about animals that will be used throughout the paper; in Section 3, we review a previous proposal that can be found in the literature; then, in Section 4, we informally define our translation of OWL DL ontologies into FOL; finally, we give some conclusions and discuss future work in Section 5.

DL Syntax	Semantics
C	$C^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$
$\top$	$\top^{\mathcal{I}} = \triangle^{\mathcal{I}}$
$\bot$	$\bot^{\mathcal{I}} = \emptyset$
$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
$\neg C$	$(\neg C)^{\mathcal{I}} = \triangle^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{\ x \mid \langle x, y\rangle \in R^{\mathcal{I}} \ \wedge\ y \in C^{\mathcal{I}}\ \}$
$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{\ x \mid \langle x, y\rangle \in R^{\mathcal{I}} \ \rightarrow\ y \in C^{\mathcal{I}}\ \}$
o	$o^{\mathcal{I}} \in \triangle^{\mathcal{I}}$

Table 1
Some basic terms about concepts and individuals

DL Syntax	Semantics
$C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$C^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
$C \equiv C_1 \sqcap \ldots \sqcap C_n$	$C^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
$C_i \sqcap C_j \equiv \bot$	$C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} = \emptyset$
$o \in C$	$o^{\mathcal{I}} \in C^{\mathcal{I}}$

Table 2
Some formulas

2 Preliminaries

OWL DL syntax and semantics are presented in [3]. In Tables 1 and 2 we recall the main aspects of OWL DL in order to understand the examples in this paper. Roughly speaking, OWL DL ontologies define use concepts (or classes, denoted by C), properties (relations or roles, denoted by R) and instances (or objects). The union of all possible classes is denoted by $\triangle$, whereas $\triangle \times \triangle$ is the union of all possible relations. By default, there are two predefined classes in any OWL DL ontology, `Thing` and `Nothing` (denoted by $\top$ and $\bot$ respectively). `Thing` is superclass of every class defined in the ontology and `Nothing` has no proper subclasses.

Dealing with classes, we use the classical notation of sets: union ($\cup$), intersection ($\cap$), inclusion ($\subseteq$), set difference ($\setminus$), equality ($=$) and empty class ($\emptyset$).

2.1 An OWL Ontology: Animals

All the examples in this paper use a small ontology about animals that comes from SUMO, the IEEE *Suggested Upper Merged Ontology* (see [8]). More specifically, our ontology consists in the hierarchy of classes defined in SUMO version 1.48 [4] starting from `Animal`

[4] We choose SUMO version 1.48 because it is mapped to WordNet 1.6

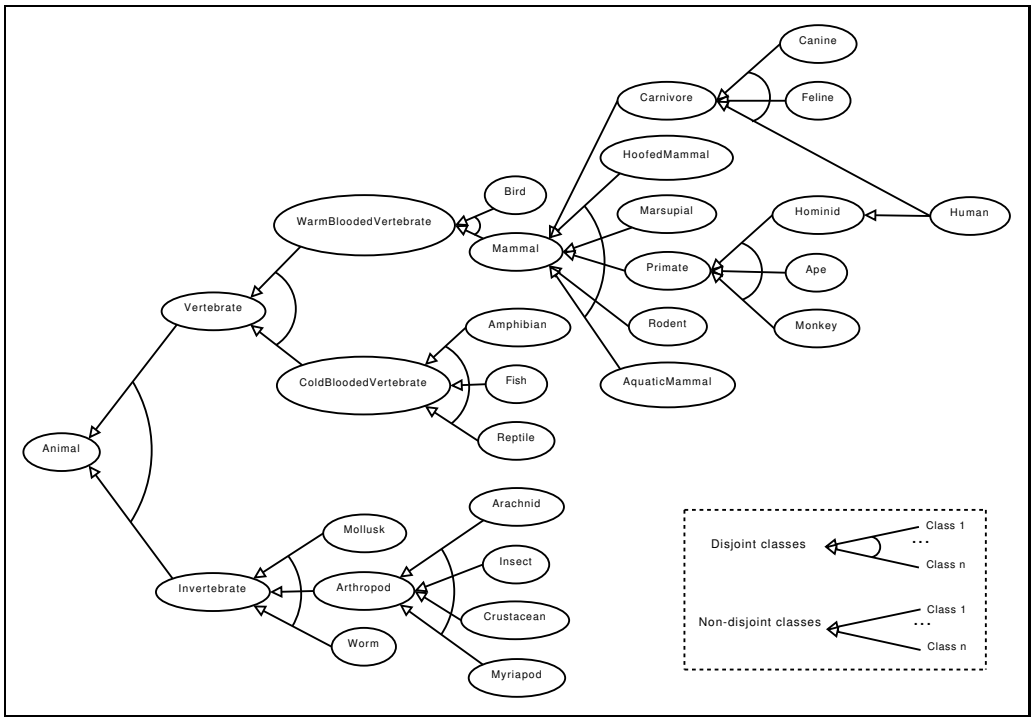

Fig. 1. Classes of the Ontology about Animals

(see Figure 1). In this hierarchy, the relations $subclass$ and $disjoint$ of SUMO have been preserved. Besides, we have included two changes in our ontology. First, `Human` is not $subclass$ of `Carnivore` in the original ontology. This change allows us to illustrate the problem of multiple inheritance. Second, we have inserted two objects as instance of each class without subclasses. To sum up, our ontology consists of 29 classes and 38 objects.

As described in Figure 1, the class `Animal` is divided into the subclasses `Vertebrate` and `Invertebrate`, which are explicitly declared to be disjoint. Both classes are also divided into several subclasses. All the subclasses of the same class are declared to be disjoint except for the class `Carnivore`, which is not disjoint with the rest of subclasses of `Mammal`. The next OWL DL sentences trivially follows from our ontology:

$$\neg(Invertebrate \sqsubseteq Bird) \qquad (Human \equiv Carnivore \sqcap Primate)$$

$$(Mammal \sqsubseteq Animal) \qquad (\bot \equiv Arthropod \sqcap Primate)$$

3 Translation Issues in Hoolet

The main proposals for using FOL theorem provers to solve OWL DL reasoning tasks are presented in [2,16]. Unfortunately, the last one is not accessible. For this reason, we only show how the first approach works, which is called *Hoolet* [2].

The Hoolet system translates an OWL DL ontology into a collection of FOL axioms on the basis of the well-known OWL DL semantics [1,4]. Then, the resulting FOL axioms are passed to the FOL theorem prover, which in this case is Vampire [10].

The translation ϕ maps OWL DL concept C and role R into unary and binary predicates $\phi_C(X)$ and $\phi_R(X, Y)$ respectively [15]. In Table 3, we describe the translation of some concept and role axioms. It is worth noting that all axioms except the last one are

DL	FOL
$C_1 \sqsubseteq C_2$	$\forall X(\phi_{C_1}(X) \rightarrow \phi_{C_2}(X))$
$C_1 \equiv (C_2 \sqcap C_3)$	$\forall X(\phi_{C_1}(X) \leftrightarrow (\phi_{C_2}(X) \wedge \phi_{C_3}(X)))$
$\bot \equiv (C_2 \sqcap C_3)$	$\forall X(\neg(\phi_{C_2}(X) \wedge \phi_{C_3}(X)))$
$R_1 \sqsubseteq R_2$	$\forall X \forall Y(\phi_{R_1}(X,Y) \rightarrow \phi_{R_2}(X,Y))$
$\top \sqsubseteq \forall R.C$	$\forall X \forall Y(\phi_R(X,Y) \rightarrow \phi_{C_2}(Y))$
$C_1 \equiv (C_2 \sqcap \exists R.C_3)$	$\forall X(\phi_{C_1}(X) \leftrightarrow [\phi_{C_2}(X) \wedge \exists Y(\phi_R(X,Y) \wedge \phi_{C_3}(Y))])$

Table 3
Translation of OWL DL Axioms in Hoolet

Horn clauses. Generally a complete translation from OWL DL into FOL obtains non-Horn clauses.

With respect to instances, the Hoolet system adds a different constant c_i for each instance i occurring in the OWL DL ontology. The problem with this is that most popular and successful theorem provers such as Vampire [10], which is the one used by Hoolet, do not implement the unique name assumption. That is, theorem provers do not assume that two different constants represent different objects. Thus, for every pair of instances i and j represented by the constants c_i and c_j that are declared to be distinct, Hoolet adds $c_i \neq c_j$ as an axiom to the translation.

However, the above information can be sometimes inferred from the remaining axioms of the ontology. That is, if c_i and c_j are instances of two disjoint classes, then Hoolet can infer that c_i and c_j are distinct.

Finally, Hoolet relates instances and classes using $\phi_C(c_i)$ as assertions when i is an instance of C.

Regarding our ontology about animals, the Hoolet system models it as follows:

Classes of Animal

$\forall X : Vertebrate(X) \rightarrow Animal(X)$

$\forall X : WarmBloodedVertebrate(X) \rightarrow Vertebrate(X)$

$\forall X : ColdBloodedVertebrate(X) \rightarrow Vertebrate(X)$

$\forall X : Fish(X) \rightarrow ColdBloodedVertebrate(X)$

...

Classes *Nothing* ($\bot$) and *Thing* ($\top$)

$\forall X : Animal(X) \rightarrow Thing(X)$

$\forall X : \neg Nothing(X)$

Disjoint classes

$\forall X : \neg(Vertebrate(X) \wedge Invertebrate(X))$

$\forall X : \neg(WarmBloodedVertebrate(X) \wedge ColdBloodedVertebrate(X))$

...

Regarding instances, our ontology includes two instances for each class without subclasses. For example, `Tuna` and `Shark` are declared as instances of `Fish`. The Hoolet

system translates instances as follows:

Instances of Animal

$$Fish(Tuna) \qquad Fish(Shark) \qquad Tuna \neq Shark$$

Hoolet allows six kinds of queries, all of them ground (without variables):

- satisfiable: checks whether the ontology is consistent or not.
- retrieve `Class`: returns all the instances of the class. For this purpose, Hoolet checks every instance defined in the ontology.
- instance `Class Individual`: checks whether `Individual` is an instance of `Class`.
- related `Individual` Property `Individual`: checks whether both individuals are related by the property.
- same `Individual Individual`: checks whether both individuals are equal.
- different `Individual Individual`: checks whether both individuals are different.

Hoolet restricts queries to the above six kinds of sentences in order to ensure the FOL theorem prover finds an answer. However, a FOL theorem prover is able to answer more queries using Hoolet's translation. For example, we obtain the following result using Vampire:

$$\forall X : \neg(Arthropod(X) \wedge Primate(X))? \text{ Yes}$$

Furthermore, Hoolet is not able to answer some queries that most OWL DL reasoners can solve. For example, while OWL DL reasoners answer "No" to the goal

$$\exists X : (X \sqsubseteq Carnivore \wedge X \sqsubseteq Hominid)$$

Hoolet does not answer. This limitation is due to Hoolet's translation, which transforms concepts (classes) into unary predicates, restricting the power of reasoning since some classical properties of binary relations, such as transitivity or symmetry, cannot be expressed using unary predicates.

Obviously, some queries cannot be solved using Hoolet or OWL DL reasoners:

$$\forall X : \ (\ (Fish \sqsubseteq X \wedge Rodent \sqsubseteq X) \rightarrow Vertebrate \sqsubseteq X\ )$$
$$\exists X : \ (\ (X \not\equiv \bot \wedge \forall Y : (Y \not\equiv X \wedge Y \not\equiv \bot)) \rightarrow \neg(Y \sqsubseteq X)\ )$$

The first query asks whether the `Vertebrate` class is a subclass of every superclass of both `Fish` and `Rodent`. The last one checks whether there exists a class without proper subclasses. Regarding our ontology about animals, such a class could be `Bird`.

Being able to solve the above queries (and any non-OWL DL query) is the aim of the translation described in the next section.

4 Our Translation Proposal

For the translation of OWL DL ontologies, we use two main classes relations: subclass ($\sqsubseteq$) and disjoint ($\diamond$). The definition of the disjoint relation is provided in Table 4. Besides, we also consider the relation instance ($\in$). All these relations are binary in order to overcome the main limitations of Hoolet. The remaining properties in an OWL DL ontology are not still translated by our system. However, the translation of these missing properties is very similar to the treatment of instances.

$\diamond$	$\Phi_\diamond$
Definition	$\forall X \forall Y\ (\ \Phi_\diamond(X,Y)\ \leftrightarrow\ \forall I \neg(\Phi_\in(I,X) \wedge \Phi_\in(I,Y))\ )$

Table 4
Definition of the Disjoint Relation

In our approach, the axiom of Table 4 will be common to any OWL DL ontology. Then, depending on the particular ontology, our system has to add new axioms, as defined in Table 5 regarding classes. For the description of the translation, we use the auxiliary symbol $\sqsubset$ in expressions of the form $C \sqsubset D$, which denotes that C is a proper and direct subclass of the class D.

OWL DL Ontology	FOL Translation
$\{\ C \mid C \sqsubset D\ \} = \{C_1, \ldots, C_n\}$	$\forall X\ (\ \Phi_\sqsubseteq(X,D) \leftrightarrow (X = D \vee \Phi_\sqsubseteq(X,C_1) \vee \ldots \vee \Phi_\sqsubseteq(X,C_n))\ )$
$\{\ C \mid D \sqsubset C\ \} = \{C_1, \ldots, C_n\}$	$\forall X\ (\ \Phi_\sqsubseteq(D,X) \leftrightarrow (X = D \vee \Phi_\sqsubseteq(C_1,X) \vee \ldots \vee \Phi_\sqsubseteq(C_n,X))\ )$
$\bot \equiv (D \sqcap C)$	$\forall I\ (\ \neg(\Phi_\in(I,D) \wedge \Phi_\in(I,C))\ )$

Table 5
FOL Translation of OWL DL Ontologies: Classes

Following this translation, our ontology about animals is modeled as follows:

Classes of Animal

$$\forall X : \Phi_\sqsubseteq(X, Animal) \leftrightarrow (\ X = Animal\ \vee\ \Phi_\sqsubseteq(X, Vertebrate) \vee \Phi_\sqsubseteq(X, Invertebrate)\ )$$
$$\forall X : \Phi_\sqsubseteq(Animal, X) \leftrightarrow X = Animal$$
$$\forall X : \Phi_\sqsubseteq(X, Vertebrate) \leftrightarrow (\ X = Vertebrate \vee \Phi_\sqsubseteq(X, ColdBloodedVertebrate) \vee \Phi_\sqsubseteq(X, WarmBloodedVertebrate)\ )$$
$$\forall X : \Phi_\sqsubseteq(Vertebrate, X) \leftrightarrow (\ X = Vertebrate\ \vee\ \Phi_\sqsubseteq(Animal, X)\ )$$
$$\ldots$$
$$\forall X : \Phi_\sqsubseteq(X, Bird)) \leftrightarrow X = Bird$$
$$\ldots$$

Disjoint classes

$$\forall I : (\ \neg[\ \Phi_\in(I, Vertebrate) \wedge \Phi_\in(I, Invertebrate)\ ]\ )$$
$$\ldots$$
$$\forall I : (\ \neg[\ \Phi_\in(I, Mammal) \wedge \Phi_\in(I, Bird)\ ]\ )$$
$$\ldots$$

With respect to objects, the translation is described in Table 6. Our translation takes into account that instance is an inheritable relation. The translation of a non-inheritable relation

produces axioms like the ones in Table 6 where right-hand subformulas in equivalences exclusively consist of equality atoms.

OWL DL Ontology	FOL Translation
$o \in D$	$\forall X\ (\ \Phi_{\in}(o, X) \leftrightarrow \Phi_{\sqsubseteq}(D, X)\ )$
$\{\ o \mid o \in D\ \} = \{o_1, \ldots, o_m\}$ $\{\ C \mid C \sqsubset D\ \} = \{C_1, \ldots, C_n\}$	$\forall I\ (\ \Phi_{\in}(I, D) \leftrightarrow (I = o_1 \vee \ldots \vee X = o_m \vee$ $\Phi_{\in}(I, C_1) \vee \ldots \vee \Phi_{\in}(I, C_n))\ )$

Table 6
FOL Translation of OWL DL Ontologies: Objects

Finally, since we are dealing with elements of different nature (that is, classes and objects), we also include type information adding two axioms. These restrict the possible values that can be assigned to a variable. Assuming that $\mathcal{C}$ and $\mathcal{O}$ respectively denotes the set of all classes and objects defined in the ontology, our translation included these two axioms

$$\forall X\ (\ \Phi_{\mathcal{C}}(X) \leftrightarrow X = \mathfrak{c}_1 \vee \ldots X = \mathfrak{c}_m\ )$$
$$\forall X\ (\ \Phi_{\mathcal{O}}(X) \leftrightarrow X = \mathfrak{o}_1 \vee \ldots X = \mathfrak{o}_n\ )$$

where $\mathcal{C} = \{\mathfrak{c}_1, \ldots, \mathfrak{c}_m\}$ and $\mathcal{O} = \{\mathfrak{o}_1, \ldots, \mathfrak{o}_n\}$.

As discussed in Section 5, the theory that results from the conjunction of all the above described axioms is sound and complete.

Once the ontology is translated into the above collection of FOL axioms, we can ask whether a FOL formula (over the signature of our theory) is a logical consequence of the theory. For this purpose, each query is typed in the classical way, restricting the type of arguments in every predicate. For example, both arguments of $\Phi_{\sqsubseteq}$ must be classes, whereas the first argument of $\Phi_{\in}$ must be an object and the second one a class. Then, every query of the form $\forall X\ \varphi$ is transformed (via the γ function) into $\forall X\ (\ \Phi_{\mathcal{C}}(X) \rightarrow \gamma(\varphi)\ )$ (resp. $\forall X\ (\ \Phi_{\mathcal{O}}(X) \rightarrow \gamma(\varphi)\ )$) if X appears in arguments restricted to classes (resp. objects) in φ. Besides, every query of the form $\exists X\ \varphi$ is replaced with $\exists X\ (\ \Phi_{\mathcal{C}}(X) \wedge \gamma(\varphi)\ )$ (resp. $\exists X\ (\ \Phi_{\mathcal{O}}(X) \wedge \gamma(\varphi)\ )$) if X occurs in arguments restricted to classes (resp. objects) in φ.

Below, in Table 7 we experimentally compare[5] the running time of both Hoolet and our system. We make ten different queries and evaluate the time used by Vampire to answer. The system is available at http://adimen.si.ehu.es/cgi-bin/Beast/index.perl

Obviously, when Hoolet is able to respond, our system takes longer than Hoolet. However, as we explain in the next section, we can answer not only the above nine queries but also any FOL query over the signature of our theory.

5 Conclusions and Future Work

So far, the results of our experiments have been very promising. Our tests prove that it is possible to use FOL theorem provers to reason with OWL DL ontologies that have been

[5] For tests, we use an Intel(R) Core(TM)2 Duo CPU E6850 @ 3.00GHz with 8Gb RAM.

FOL query	Hoolet	Our System
$\Phi_{\in}(Tuna, Animal)$	Yes (0.003 seconds)	Yes (0.025 seconds)
$\Phi_{\sqsubseteq}(Mammal, Animal)$	Yes (0.003s)	Yes (0.024s)
$\Phi_{\sqsubseteq}(Invertebrate, Bird)$	cannot prove	No (0.018s)
$\Phi_{\diamond}(Arthropod, Primate)$	Yes (0.005s)	Yes (0.052s)
$\Phi_{\diamond}(Carnivore, Primate)$	cannot prove	No (0.024s)
$\forall X((\Phi_{\sqsubseteq}(Fish, X) \wedge \Phi_{\sqsubseteq}(Rodent, X)) => \Phi_{\sqsubseteq}(Vertebrate, X))$	cannot prove	Yes (0.056s)
$\exists X \forall Y((X \neq Y) => \neg (\Phi_{\sqsubseteq}(Y, X)))$	cannot prove	Yes (0.013s)
$\forall X(\Phi_{\sqsubseteq}(X, X))$	cannot prove	Yes (0.110s)
$\forall X \forall Y \forall Z((\Phi_{\sqsubseteq}(X, Y) \wedge \Phi_{\sqsubseteq}(Y, Z)) => \Phi_{\sqsubseteq}(X, Z))$	cannot prove	Yes (0.672s)
$\forall I \forall X \forall Y((\Phi_{\in}(I, X) \wedge \Phi_{\sqsubseteq}(X, Y)) => \Phi_{\in}(I, Y))$	cannot prove	Yes (49.260s)

Table 7
Running Time

properly translated into FOL. Therefore, we think that it is reasonable to continue the work on this area.

Another remarkable conclusion of this work is that a suitable translation (or a suitable representation of knowledge) allows to overcome the limitations of current FOL theorem provers, which are not *ad hoc* tools for reasoning with ontologies. However, finding a *good* translation is not an easy task. It depends on the way in which FOL theorem provers work and, of course, on the kind of queries to solve. Our translation has taken into account that general purpose FOL theorem provers are resolution-based. However, we have not added redundant information about properties of binary relations (reflexive, transitive property, etc.), which can be inferred from the remaining axioms that result from our translation. Nevertheless, adding this redundant information could help theorem provers from solving some queries in much less time.

Further, it is worth to note that we have empirically proved [6] the complete nature of the resulting theories. More specifically, we automatically check that any ground atom (or its negation) that can be constructed using the predicates and constants from the ontology is a logical consequence of the theory. This fact ensures that, when typing queries, we can solve any of them, by running the query and its negation in parallel.

However, there are still some limitations due to the use of current FOL theorem provers. For instance, we only obtain boolean answers to existential queries, instead of providing the values that satisfy the query. Another restriction is the lack of goal-oriented proof-

[6] The execution time of the completeness proof is about 15 minutes.

search techniques, which prevents the use of FOL theorem provers for reasoning on large ontologies. We try to overcome this last problem by pre-processing queries in order to remove non-relevant axioms for each particular query.

Our most immediate future aim is to continue expanding the translation of OWL DL into FOL until covering OWL DL syntax in its entirety. For this purpose, first we have to translate OWL DL properties into FOL. In our opinion, the translation of properties is very similar to the one of instances. Hence, we expect to fulfill this task soon.

Once finishing the treatment of OWL DL ontologies, we also plan to reason with more expressive ontologies, as discussed in [6]. Anyway, the problem of reasoning on ontologies like SUMO [8] using FOL theorem provers is still open.

References

[1] Baader, F., D. Calvanese, D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, editors, "The Description Logic Handbook: Theory, Implementation, and Applications," Cambridge University Press, 2003.

[2] Bechhofer, S. and I. Horrocks, *Hoolet: An OWL reasoner with support for rules* (2004).
URL http://owl.man.ac.uk/hoolet/

[3] Bechhofer, S., F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider and L. A. Stein, *OWL web ontology language reference. W3C recommendation* (2004).
URL http://www.w3.org/TR/owl-ref

[4] Borgida, A., *On the relative expressiveness of description logics and predicate logics*, Artificial Intelligence **82** (1996), pp. 353–367.

[5] Horrocks, I. and P. F. Patel-Schneider, *Reducing OWL entailment to description logic satisfiability*, Journal of Web Semantics **1** (2004), pp. 345–357.

[6] Horrocks, I. and A. Voronkov, *Reasoning support for expressive ontology languages using a theorem prover*, in: J. Dix and S. J. Hegner, editors, *Foundations of Information and Knowledge Systems (FoIKS 2006)*, Lecture Notes in Computer Science **3861** (2006), pp. 201–218.

[7] Knublauch, H., R. W. Fergerson, N. F. Noy and M. A. Musen, *The Protégé OWL plugin: An open development environment for semantic web applications*, in: S. A. McIlraith, D. Plexousakis and F. van Harmelen, editors, *Proceedings of the International Semantic Web Conference (ISWC 2004)*, Lecture Notes in Computer Science **3298** (2004), pp. 229–243.

[8] Niles, I. and A. Pease, *Towards a standard upper ontology*, in: *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS '01)* (2001), pp. 2–9.

[9] Pelletier, F. J., G. Sutcliffe and C. B. Suttner, *The development of CASC*, AI Communications **15** (2002), pp. 79–90.

[10] Riazanov, A. and A. Voronkov, *The design and implementation of VAMPIRE*, AI Communications **15** (2002), pp. 91–110.

[11] Schulz, S., *E – a brainiac theorem prover*, Journal of AI Communications **15** (2002), pp. 111–126.

[12] Sirin, E., B. Parsia, B. C. Grau, A. Kalyanpur and Y. Katz, *Pellet: A practical OWL-DL reasoner*, Journal of Web Semantics **5** (2007), pp. 51–53.

[13] Sutcliffe, G. and C. B. Suttner, *The state of CASC*, AI Communications **19** (2006), pp. 35–48.

[14] Tsarkov, D. and I. Horrocks, *FaCT++ description logic reasoner: System description*, in: U. Furbach and N. Shankar, editors, *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2006)*, Lecture Notes in Computer Science **4130** (2006), pp. 292–297.

[15] Tsarkov, D., A. Riazanov, S. Bechhofer and I. Horrocks, *Using Vampire to reason with OWL*, in: S. A. McIlraith, D. Plexousakis and F. van Harmelen, editors, *Proceedings of the International Semantic Web Conference (ISWC 2004)*, Lecture Notes in Computer Science **3298** (2004), pp. 471–485.

[16] Zhang, Z. and J. A. Miller, *Ontology query languages for the semantic web: A performance evaluation*, Technical Report UGA-CS-LSDIS-TR-05-011, Department of Computer Science, University of Georgia, Athens (2005).