On the Utility and Feasibility of Reasoning with Undecidable Semantic Web Formalisms

Technical Report 3016, Institute AIFB, KIT

Sebastian Rudolph and Michael Schneider

Institute AIFB, Karlsruhe Institute of Technology, DE
rudolph@kit.edu
FZI Research Center for Information Technology, Karlsruhe, DE
schneid@fzi.de

Abstract. Semantic Web knowledge representation standards such as RDF and OWL have gained momentum in the last years and are widely applied today. In the course of the standardization process of these and other knowledge representation formalisms, decidability of logical entailment has often been advocated as a central design criterion. On the other hand, restricting to decidable formalisms inevitably comes with constraints in terms of modeling power. Therefore, in this paper, we examine the requirement of decidability and weigh its importance in different scenarios. Subsequently, we discuss a way to establish incomplete – yet useful – reasoning support for undecidable formalisms by deploying machinery from the successful domain of theorem proving in first-order predicate logic. While elaborating on the undecidable variants of the ontology language OWL 2 as our primary examples, we argue that this approach could likewise serve as a role model for knowledge representation formalisms from the Conceptual Structures community.

1 Introduction

Today, the Semantic Web serves as the primary testbed for practical application of knowledge representation. A plethora of formalisms for representing and reasoning with Web knowledge has been designed and standardized under the auspices of the World Wide Web Consortium (W3C). While the early days of this endeavor saw ad-hoc and semantically underspecified approaches, interoperability requirements enforced their evolution into mature logical languages with clearly specified formal semantics. In the process of defining more and more expressive such formalisms, an often-debated requirement is decidability of logical entailment, i.e. the principled existence of an algorithm that decides whether a body of knowledge has a certain proposition as a consequence. While it goes without saying that such an algorithm is clearly useful for all kind of querying or

knowledge management tasks, results established back in the 1930s show that this property does not hold for all types of logics [6, 24]. In particular, in many expressive knowledge representation formalisms (most notably first-order predicate logic), entailment is undecidable.

Hence, whenever a knowledge representation formalism is to be designed, the trade-off between decidability and expressivity has to be taken into account. An examination of the Semantic Web languages hitherto standardized by the W3C yields a mixed picture in that respect: logical entailment in the basic data description language RDF [13] and its lightweight terminological extension RDF Schema [4] is decidable (although already NP-complete in both cases). Within the OWL 2 language family [17], only the most expressive variant OWL 2 Full [20] is undecidable, whereas OWL 2 DL [16, 15] as well as its specified sublanguages (called tractable profiles) OWL 2 EL, OWL 2 QL, OWL 2 RL [14] are decidable (the latter three even in polynomial time). On the other hand, for the rule interchange format RIF [12], only the very elementary core dialect RIF-Core [2] is decidable whereas already the basic logic dialect RIF-BLD [3] – and hence every prospective extension of it – turns out to be undecidable.

This small survey already shows that decidability is far from being a common feature of the standardized Semantic Web languages. However, extensive reasoning and knowledge engineering support is currently only available for the decidable languages in the form of RDF(S) triple stores or OWL 2 DL reasoners hinting at a clear practitioners' focus on these languages.

In this paper, we argue that inferencing support is important and feasible also in formalisms which are undecidable and we provide an outlook how this can be achieved, referring to our recent work on reasoning in undecidable Semantic Web languages as a showcase. We proceed as follows: Section 2 will remind the reader of the important notions from theoretical computer science. Section 3 proposes a schematic classification of inferencing algorithms by their practical usefulness. Section 4 distinguishes cases where decidability is crucial to enable "failsafe" reasoning from cases where it may make sense to trade decidability for expressivity. After these general considerations, we turn to variants of OWL to demonstrate ways to provide reasoning support for undecidable Semantic Web formalisms. To this end, Section 5 gives an overview of OWL syntaxes and the associated semantics. Section 6 shows two different ways of translating OWL reasoning problems into first-order logic and Section 7 briefly reports on our recent work of employing FOL reasoners in that context. In Section 8, we discuss ramifications of our ideas for Common Logic. Section 9 concludes.

Appendix A provides concrete examples of reasoning in the undecidable languages treated in this paper.

2 Recap: Decidability and Semidecidability

Let us first recap some basic notions from theoretical computer science which are essential for our considerations. From an abstract viewpoint, a logic is just a (possibly infinite) set of *sentences*. The *syntax* of the logic defines how these sentences look like. The *semantics* of the logic is captured by an *entailment relation* $\models$ between sets of sentences Φ and sentences φ of the logic. $\Phi \models \varphi$ then means that Φ logically entails φ or that φ is a logical consequence of Φ. Usually, the logical entailment relation is defined in a model-theoretic way.

A logic is said to have a *decidable* entailment problem (often this wording is shortened as to calling the logic itself decidable – we will adopt this common practice in the following) if there is an algorithm which, taking as an input a finite set $\Phi = \{\phi_1, \ldots, \phi_n\}$ of sentences of that logic and a further sentence φ, always terminates and provides the output *YES* iff $\Phi \models \varphi$ or *NO* iff $\Phi \not\models \varphi$. As a standard example for a decidable logic, propositional logic is often mentioned, a straightforward decision procedure being based on truth tables. However, there are decidable logics of much higher expressivity, e.g., the guarded fragment of first-order logic [1].

A logic is said to have a *semidecidable* entailment problem (also here, this can be abbreviated by calling the logic itself semidecidable) if there exists an algorithm that, again given Φ and φ as input, terminates and provides the output *YES* iff $\Phi \models \varphi$, but may not terminate otherwise. Consequently, such an algorithm is complete in the following sense: every consequence will eventually be identified as such. However the algorithm cannot be used to get a guarantee that a certain sentence is *not* a consequence. Clearly, every decidable logic is also semidecidable, yet, the converse does not hold. The prototypical example for a logic that is semidecidable but not decidable is first-oder predicate logic (FOL). While undecidability of FOL can be shown by encoding the halting problem of a Turing machine into a FOL entailment problem, its semidecidability is a consequence from the fact that there exists a sound and complete deduction calculus for FOL [7], hence every consequence can be found in finite time by a breadth-first search in the space of all proofs w.r.t. that deduction calculus. Clearly, today's first-order theorem provers use much

more elaborated and goal-directed strategies to find a proof of a given entailment.

Obviously, in semi-decidable logics, the critical task which cannot be completely solved, is to detect the non-entailment $\Phi \not\models \varphi$. In model-theoretically defined logics such as FOL, $\Phi \not\models \varphi$ means that there exists a model $\mathcal{M}$ of Φ that is not a model of φ. In other words, finding such a model means proving the above non-entailment. Indeed, there are rather effective (yet incomplete) FOL model finders available dedicated to this purpose. For straightforward reasons, most of these model finders focus on finite models. In fact, if there is a finite model with the wanted property, it is always possible to find it (due to the reason that the set of finite models is enumerable and first-order model checking is easy). Hence, the case which is intrinsically hard to automatically detect is when $\Phi \not\models \varphi$ but every model of Φ that is not a model of φ has *infinite* size. While seemingly exotic at first sight, such cases exist and are not very hard to construct. An example for such a situation is the question whether $\varphi = \exists x.(p(x,x))$ is a logical consequence of $\Phi = \{\varphi_1, \varphi_2\}$ with $\varphi_1 = \forall x.\exists y.(p(x,y))$ and $\varphi_2 = \forall x \forall y \forall z.(p(x,y) \wedge p(y,z) \rightarrow p(x,z))$. In this example, φ_1 enforces that in every model of Φ every element must be in a p-relationship to something, whereas φ_2 requires that p must be interpreted by a transitive relation. A side effect of p's transitivity is that whenever a model contains a p-cycle, all the elements in that cycle are p-related to themselves which makes φ satisfied. Therefore, any model of Φ that does not satisfy φ must be p-cycle-free which is only possible if the model is infinite. The problem with infinite models is that, even if one has a method to represent and refer to them somehow, the set of all infinite models cannot fully be enumerated. Hence, whatever enumeration strategy is used, it will only cover a strict subset of all possible models.

3 A Classification of Decision Procedures by Usefulness

We will now take a closer look on the question how useful a sound and complete decision algorithm may be in practice. For the sake of better presentation, assume that we consider not all the (infinitely many) possible inputs to the algorithm but only the finitely many (denoted by the set P) below a fixed size s. Let us furthermore make the very simplifying assumption that every entailment problem in P is equally "important" in the sense that it will be posed to our reasoning algorithm with the same probability as the other problems. Now, a decision algorithm $\mathcal{A}$ comes with the guarantee, that it terminates on each of the inputs after finite

time, hence it gives rise to a function $\mathsf{runtime}_{\mathcal{A}} : P \to \mathbb{R}^+$ assigning to each of the inputs the corresponding (finite) runtime of the algorithm. Now, the algorithm can be described by a *characteristic curve* assigning to a time span Δt the fraction of elements from P on which $\mathcal{A}$ terminates after time less or equal to Δt. Formally, this function $\mathsf{char}_{\mathcal{A}} : \mathbb{R}^+ \to [0, 1]$ would be defined by

$$\mathsf{char}_{\mathcal{A}}(\Delta t) = \frac{\{p \in P \mid \mathsf{runtime}_{\mathcal{A}}(p) \leq \Delta t\}}{|P|}.$$

Figure 1 schematically displays characteristic curves which may be encountered for complete decision procedures. As a common feature, note that the curves are always monotonic by definition. Moreover, every such curve will hit the 100% line at some time point due to the facts that we have a complete decision algorithm and P is finite.

We now assume that the decision algorithm is to be employed in a practical scenario, which gives rise to the following specific figures:

- a maximal time span that is worth to be spent on the computation of an answer to an entailment problem of size s, referred to as *acceptable waiting time*;
- a ratio characterizing the probability of getting an answer below which a use of the algorithm will be considered worthless, called the *perceived added value threshold*; and
- a ratio characterizing the probability of getting an answer above which the algorithm can be considered practically reliable, called the *acceptable reliability threshold*.

Figure 1 also depicts these values, according to which the four schematic characteristic curves can now be coarsely distinguished in terms of practical usefulness.

- In the ideal case, the maximal runtime of the algorithm is smaller than the acceptable waiting time. Then every size s problem is guaranteed to be decided within the available time span, which allows for calling the algorithm *failsafe*.
- If this guarantee cannot be given, yet the probability that a solution will be obtained in the available time lies above the acceptable reliability threshold, the algorithm can still be said to be (practically) *reliable* and may be used within regular and automated knowledge management work flows. Then the rare cases not being covered could be dealt with by a kind of controlled exception handling mechanism.

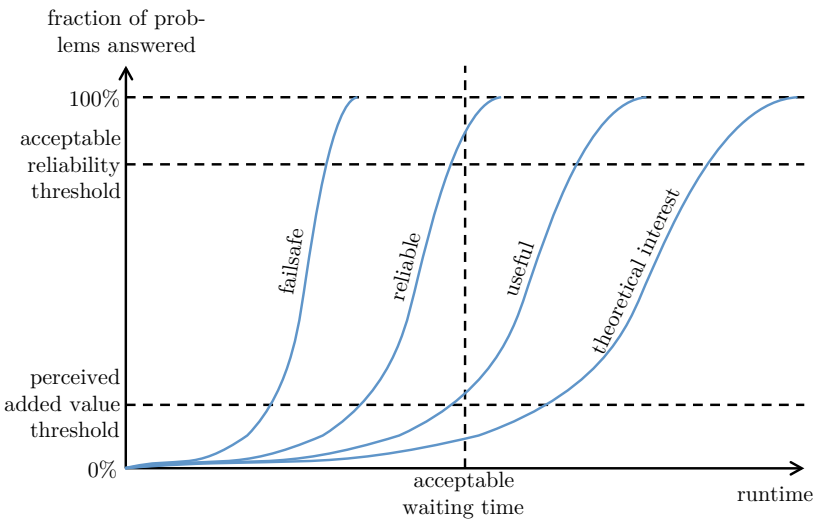

Fig. 1. Schematic representation of different characteristic curves for complete decision algorithms.

- If the expected frequency of termination within the available time span is below the acceptable reliability threshold but above the perceived added value threshold, the algorithm's output should be perceived as nice-to-have additional information which may be taken into account if available. The overall work flow in which it is to be used should not crucially rely on this, yet we can still see that the algorithm may be rightfully called *useful*.
- Finally, if the ratio of the timely obtainable answers is even below the perceived added value threshold, the algorithm is of little or no practical value. Note however, that the existence of a complete decision algorithm – even if it happens to lie within this class – is still a research question worthwhile pursuing since optimizations and hardware improvements may well turn such an algorithm into something practically useful. Conversely, the proven non-existence of a decision algorithm may prevent many ambitious researchers from vainly trying to establish one. This justifies to at least characterize this type of algorithm as of *theoretical interest*. For a prototypical example of this kind see [19].

Now, turning our attention to incomplete decision algorithms, we can assign to them characteristic curves in the same way as to complete ones

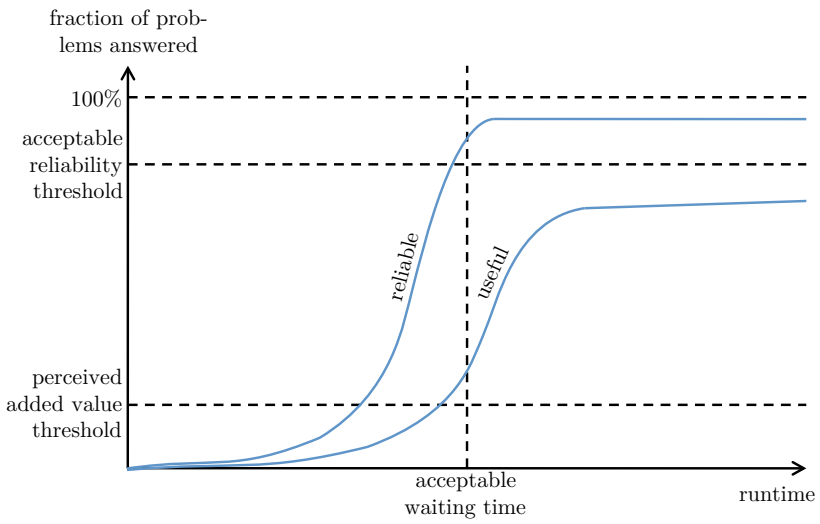

Fig. 2. Sketches of characteristic curves for incomplete decision procedures of practical interest.

(extending the range of $\mathsf{runtime}_{\mathcal{A}}$ to $\mathbb{R}^+ \cup \{\infty\}$). The only difference here is, that the curve will not hit the 100% line. Nevertheless, as Fig. 2 illustrates, there may be incomplete algorithms still satisfying the usefulness or even the reliability criterion defined above leading to the conclusion that there may be cases where the practically relevant behavior of incomplete decision algorithms is just as good as that of complete ones, the notable exception being that no failsafe behavior can be obtained. It should be noted here that inferencing in expressive decidable formalisms comes with high worst-case complexities (normally ExpTime and higher, for instance N2ExpTime for OWL 2 DL) which implies that there are "malicious" inputs of already small size for which the runtime exceeds any reasonable waiting time. Although the runtime on average cases is usually much better, this fact normally vitiates failsafety guarantees. Of course, the case is different for so-called tractable languages which are of time-polynomial or even lower complexity.

4 On the Importance of Failsafe Decision Procedures

Having identified the possible existence of a failsafe decision procedure as the only principled practical advantage that the use of a decidable formalism may have, let us now investigate under which circumstances

such a procedure is strictly needed and in what cases it is dispensable. In the following, we will identify general criteria for this assessment.

4.1 Where Failsafe Decidability is Crucial

Failsafe decidability is important in situations where automated reasoning is a central part of the functionality of a knowledge-based system and the questions posed to the system are normally "yes-or-no" questions of a symmetric form, the answers to which are to be used to trigger different follow-up actions in a highly or purely automated system ("if yes, then do this, otherwise do that").

One important case of this is when entailment checks are to be carried out in "closed-world situations," that is, when one can assume that all necessary information about a state of affairs has been collected in a knowledge base such that the non-entailment of a queried proposition justifies the assumption that the situation expressed by that proposition does indeed not hold. Under these circumstances, both possible answers to an entailment check provide information about the real state of affairs.

In other cases, the entailment check may be aimed at finding out facts about the considered knowledge base itself, instead of the real-world domain that it describes. In fact, this can be seen as a particular closed-world situation. As an example for this, consider the reasoning task of classification, i.e. the computation of a conceptual hierarchy. Here, the typical reasoning task to be performed is to answer the question "are two given classes in a subsumption relationship, or not?" This is a common task in today's ontology management systems, useful both for supporting human ontology engineers and speeding up further reasoning tasks.

4.2 Where Failsafe Decidability is Dispensable

Yet, there are also scenarios, where failsafe decision procedures for logical entailment seem to be of less importance.

First of all, this is the case when the emphasis of the usage of knowledge representation is on modeling rather than on automated inferencing. In fact, there will often be situations where logical languages are just used for noting down specifications in an unambiguous way and probably making use of available tool support for modeling and navigating these specifications. Clearly, in these cases, no reasoning support whatsoever is required, let alone failsafe decision procedures.

In other cases, reasoning may still be required, yet the available knowledge is assumed to be sound but incomplete w.r.t. the described domain,

i.e. we have an "open-world situation." Then, non-entailments cannot be conceived as guarantees for non-validity in the domain, as opposed to entailments, which (given that the knowledge base is sound) ensure validity. As a consequence thereof, non-entailment information is of less immediate value and an entailment checker just notifying the user of established entailments may be sufficient.

A more concrete case where failsafe decidability will often not be a strict requirement is human modeling support that will alarm the knowledge engineer upon the detection of inconsistencies in the knowledge base. (Note that detecting if a knowledge base KB is inconsistent is equivalent to checking the entailment $KB \models false$.) In practical ontology engineering, one certainly wants to make sure that a created ontology is free of semantic errors, and therefore good reasoning support for the detection of inconsistencies should be available. For many application domains, it will then be sufficient if such inconsistency checking does not find an issue after some considerable time of searching, while true consistency confirmation will only be nice to have.

Query answering scenarios represent a further type of setting normally not requiring failsafe decision procedures. When doing query answering, as e.g. in SPARQL, one has, in the simplest case, one or more axioms containing variables where otherwise individuals or class expressions would occur. One asks for all solutions that semantically follow from the queried knowledge base and match the query pattern. In this scenario, one is not interested in non-solutions, but in an *enumeration* of solutions. Therefore, what one needs is an enumeration algorithm, which does not really require a complete decision procedure. Also, from the practical viewpoint, in many applications such as search, completeness of the presented set of solutions is less important than when just one entailment is to be checked. What normally matters is that *enough* solutions are provided and that *relevant* solutions are among them, relevance being a measure that has to be defined for the specific purpose.

5 OWL, Syntactic and Semantic Variants

After these abstract considerations we will turn to OWL 2 as a specific knowledge representation formalism where the aforementioned issues are of particular interest, since both decidable and undecidable versions exist as well as well-investigated reasoning approaches for either. We will particularly focus on approaches for reasoning in undecidable formalisms.

First, let us recap the main aspects of syntax and semantics of OWL 2 to the degree needed for our considerations. When dealing with the Web Ontology Language OWL, one has to distinguish between two representation strategies which are interrelated but not fully interchangeable. The so called *functional syntax* [16] emphasizes the formula structure of what is said in the logic. As an example, consider the fact that an individual characterized as "happy cat owner" indeed owns some cat and everything he or she cares for is healthy. Expressed in OWL functional syntax, this statement would look as follows:

```
SubClassOf(
   ex:HappyCatOwner
   ObjectIntersectionOf(
     ObjectSomeValuesFrom( ex:owns ex:Cat )
     ObjectAllValuesFrom( ex:caresFor ex:Healthy ) ) )
```

On the other hand, there is the RDF syntax [18] which expresses all information in a graph (which in turn is usually encoded as a set of vertice-edge-vertice triples labeled with so-called Uniform Resource Identifiers, short: URIs). The above proposition in RDF representation would

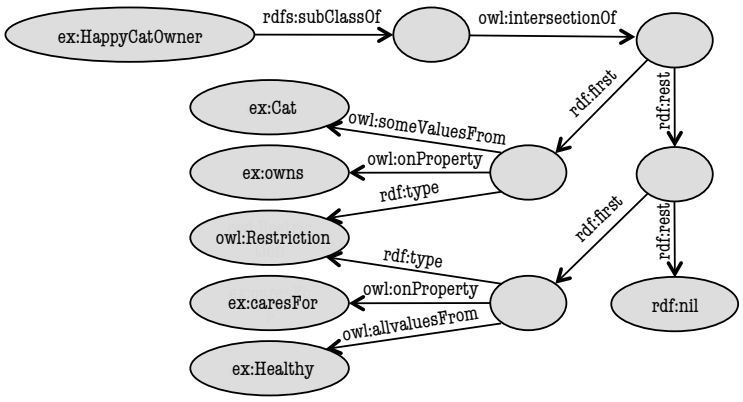

Fig. 3. RDF representation as graph.

look as displayed in Fig. 3. As opposed to the XML-based syntax often used for machine processing, the so-called Turtle syntax better reflects the

triple structure, yet allows for abbreviations for structural bnodes[1] and list structures. The Turtle representation of the RDF graph from Fig. 3 would look as follows:

```
ex:HappyCatOwner rdfs:subClassOf
   [ owl:intersectionOf
     ( [ rdf:type owl:Restriction ;
         owl:onProperty ex:owns ;
         owl:someValuesFrom ex:Cat ]
       [ rdf:type owl:Restriction ;
         owl:onProperty ex:caresFor ;
         owl:allValuesFrom ex:Healthy ] ) ] .
```

Clearly, every functional syntax ontology description can be transformed into RDF representation. The converse is, however, not always the case. For all ontologies expressible in functional syntax, the specification provides the so-called *direct semantics* [15]. This semantics is very close to the extensional semantics commonly used in description logics: an interpretation is defined by choosing a set as domain, URIs denoting individuals are mapped to elements of that domain, class URIs to subsets and property URIs to sets of pairs of domain elements. It is noteworthy that the class of ontologies expressible via the functional syntax is not decidable, but only a subclass of it where further, so-called *global restrictions* apply. It is this decidable class which is referred to as OWL 2 DL and for which comprehensive tool support is available both in terms of ontology management, infrastructure and inferencing (software implementing decision procedures for entailment are typically called *reasoners*).

On the other hand, there is a formal semantics applicable to arbitrary RDF graphs. This so-called *RDF-based semantics* [20] is more in the spirit of the semantics of RDF(S): all URIs are mapped to domain elements in the first place and might be further interpreted through an extension function.

The two different semantics are related. A *correspondence theorem* [20] ensures that, given certain conditions which are easy to establish, the direct semantics on an ontology in functional syntax (taking into account the global restrictions of OWL 2 DL) will only provide conclusions that the RDF-based semantics provides from the ontology's RDF counterpart as well. However, the RDF-based semantics will often provide additional conclusions that are not provided by the direct semantics, for instance

[1] Bnodes, also called blank nodes, are unlabeled vertices in the RDF graph representation and often used as auxiliary elements to encode complex structures.

conclusions based on metamodeling. The difference between the two semantics becomes significant for those ontologies that are still covered by the direct semantics but are not OWL 2 DL ontologies, i.e. ontologies that are beyond the scope of the correspondence theorem. Furthermore, there are ontologies to which only the RDF-based semantics applies but not the direct semantics, since not every RDF graph can be translated into an ontology in functional syntax. Figure 4 summarizes the relationships just stated.

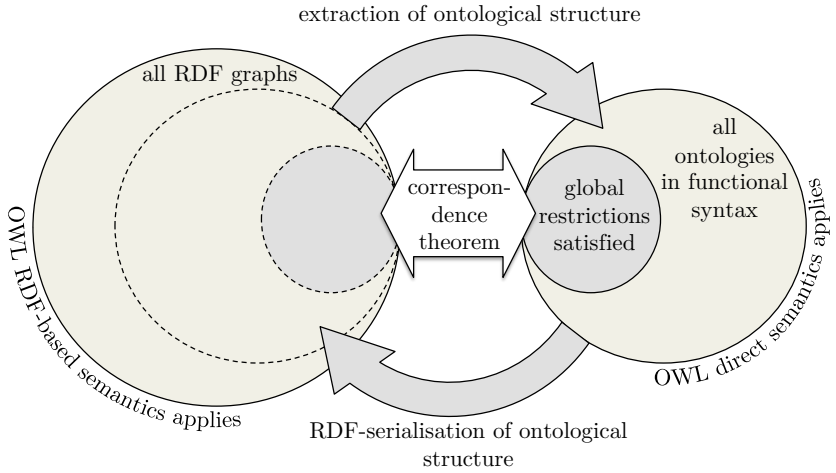

Fig. 4. Syntaxes and semantics for OWL 2.

6 FOL-Empowered OWL Reasoning beyond Decidability

As stated before, for the decidable fraction of OWL 2, decision procedures have been implemented and are widely used today. Here, we will focus on the problem of providing practical inferencing support for the more expressive yet undecidable versions of OWL. Thereby, we want to question an attitude sometimes encountered in the OWL community according to which the quest for automated inferencing beyond decidability is futile.

It seems in place to take a look at the prototypical example of undecidability: first-order predicate logic. Despite the fact that its undecidability was proven way before the wide-spread adoption of computers, automated first-order logic reasoning has always been a focus of interest and is mean-

while well-established also in practical applications such as hard- and software verification [26]. There are many FOL theorem provers which are able to find out when a set of FOL formulas is unsatisfiable or implies another set of formulas. Even model-finding is well supported, although typically restricted to finding finite models.[2]

In view of the success of FOL-based inferencing, which also substantiates our claim that reasoning in undecidable formalisms can be useful, it seems tempting to harness the comprehensive theoretical work and highly optimized implementations originating from that line of research for coping with inferencing problems in the undecidable variants of OWL. As it turns out, OWL inferencing with respect to both syntaxes and according semantics admits translation into FOL entailment problems. In the following, we will describe these embeddings in a bit more detail.

6.1 Translating Direct Semantics Reasoning into FOL

Given an ontology in functional syntax, the translation into FOL can be performed along the lines of the standard translation of description logics into FOL as, e.g., described in [9]. For instance, the ontology axiom presented in Section 5 would be translated into the following FOL sentence:

$$\forall x.\mathrm{HappyCatOwner}(x) \rightarrow \\ \exists y.(\mathrm{owns}(x,y) \wedge \mathrm{Cat}(y)) \wedge \forall z.(\mathrm{caresFor}(x,z) \rightarrow \mathrm{Healthy}(z))$$

We see that URIs referring to classes are translated into unary predicates while those denoting properties are mapped to binary predicates. The existential and the universal property restrictions have been translated into an existentially and a universally quantified subformula, respectively. The two subformulas are connected by a conjunction symbol, which is the result of translating the intersection class expression. Finally, the outermost class subsumption axiom has been translated into a universally quantified implication formula.

After this transformation, reasoning can be performed by means of FOL reasoners: FOL theorem provers can be used to find entailments and to detect inconsistent ontologies, whereas FOL model finders can be used to detect non-entailment and to confirm consistency of an ontology. In fact this approach is not new but has already been demonstrated in [23].

[2] For a comprehensive collection of online-testable state-of-the-art provers and model finders, we refer the reader to `http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP`.

In general, one can use this approach to embed OWL 2 DL into arbitrary subsets of FOL. This includes reasoning in the OWL 2 direct semantics beyond OWL 2 DL, i.e., reasoning with ontologies that are given in the functional syntax but are not constrained by any of the global restrictions of OWL 2 DL. This relaxation allows, for example, to recognize circular relationships such as cyclic chemical molecules, a feature that has often been asked for but is not available in OWL 2 DL due to its syntactic restrictiveness (see e.g. [25]). This strategy further covers many well-known undecidable extensions of OWL such as the semantic web rules language SWRL [10] as well as the combination of the rule interchange dialect RIF BLD with OWL described in [5].

6.2 Translating RDF-Based Semantics Reasoning into FOL

Under the RDF-based semantics, a translation into FOL is also possible but works differently than for the direct semantics. The RDF graph representation of the example ontology is translated into a conjunction of ternary atomic FOL formulas, which correspond to the different RDF triples in the RDF graph:

$$
\begin{aligned}
&\exists b_0, b_1, b_2, b_3, b_4.(\\
&\quad \text{iext}(\texttt{rdfs:subClassOf}, \texttt{ex:HappyCatOwner}, b_0) \\
&\wedge\ \text{iext}(\texttt{owl:intersectionOf}, b_0, b_1) \\
&\wedge\ \text{iext}(\texttt{rdf:first}, b_1, b_3) \\
&\wedge\ \text{iext}(\texttt{rdf:rest}, b_1, b_2) \\
&\wedge\ \text{iext}(\texttt{rdf:first}, b_2, b_4) \\
&\wedge\ \text{iext}(\texttt{rdf:rest}, b_2, \texttt{rdf:nil}) \\
&\wedge\ \text{iext}(\texttt{rdf:type}, b_3, \texttt{owl:Restriction}) \\
&\wedge\ \text{iext}(\texttt{owl:onProperty}, b_3, \texttt{ex:owns}) \\
&\wedge\ \text{iext}(\texttt{owl:someValuesFrom}, b_3, \texttt{ex:Cat}) \\
&\wedge\ \text{iext}(\texttt{rdf:type}, b_4, \texttt{owl:Restriction}) \\
&\wedge\ \text{iext}(\texttt{owl:onProperty}, b_4, \texttt{ex:caresFor}) \\
&\wedge\ \text{iext}(\texttt{owl:allValuesFrom}, b_4, \texttt{ex:Healthy}))
\end{aligned}
$$

All atoms are built from a single FOL predicate 'iext', which corresponds to the function 'IEXT(.)' used in the RDF-based semantics to represent *property extensions*. Terms within the atoms are either constants or existentially quantified variables, which correspond to the URIs and the blank nodes in the RDF triples, respectively.

The above formula only represents the RDF graph itself without its meaning as an OWL 2 Full ontology. The OWL 2 Full meaning of an

RDF graph is primarily defined by the collection of model-theoretic *semantic conditions* that underly the RDF-based semantics. The semantic conditions add meaning to the terms being used in the graph, such as '`rdfs:subClassOf`', and by this means put constraints on the use of the 'IEXT(.)' function. The semantic conditions have the form of first-order formulas and can therefore be directly translated into FOL formulas. For example, the FOL representation for the semantic condition about the term '`rdfs:subClassOf`' [20, Sec. 5.8] has the form:

$$\begin{aligned}\forall c_1, c_2.(\, &\mathrm{iext}(\texttt{rdfs:subClassOf}, c_1, c_2) \leftrightarrow \\ &\mathrm{iext}(\texttt{rdf:type}, c_1, \texttt{owl:Class}) \\ \wedge \, &\mathrm{iext}(\texttt{rdf:type}, c_2, \texttt{owl:Class}) \\ \wedge \, &\forall x.(\, \mathrm{iext}(\texttt{rdf:type}, x, c_1) \rightarrow \mathrm{iext}(\texttt{rdf:type}, x, c_2) \,) \,)\end{aligned}$$

The RDF-based semantics consists of several hundred semantic conditions. The meaning of the example ontology is given by the FOL translation of the actual RDF graph plus the FOL translations of all the semantic conditions of the RDF-based semantics.

While the semantic conditions of the RDF-based semantics only quantify over elements of the domain, a restricted form of higher-order logic (HOL) is provided by both the syntax and semantics of OWL 2 Full, without however the full semantic expressivity of HOL. Nevertheless, ontologies with flexible metamodeling are supported and some useful HOL-style reasoning results can be obtained.

7 Experimental Results

In previous work, we have translated the OWL 2 RDF-based semantics into a FOL theory and done a series of reasoning experiments using FOL reasoners, which generally indicate that rather complicated reasoning beyond OWL 2 DL is possible. These experiments included reasoning based on metamodeling as well as on syntactic constellations that violate the global restrictions of OWL 2 DL. The used FOL reasoners generally succeeded on most or all of the executed tests, and even often did so considerably fast, while the state-of-the-art OWL 2 DL reasoners that were used for comparison only succeeded on a small fraction of the tests. However, one often observed problem of the FOL reasoners needs to be mentioned: while they were very successful in solving complicated problems, they often showed difficulties on large input sizes. In the future, we will have to further investigate how to cope with this scalability issue. Our results have been described in [21].

In addition, we have recently conducted some initial experiments concerning reasoning in the direct semantics beyond OWL 2 DL, including positive and negative entailment checking based on cyclic relationships. Again, we have found that FOL reasoners can often quickly solve such problems, provided that the input ontologies are of reasonable size.

8 Come on, Logic!

Common Logic (CL) [11] has been proposed as a unifying approach to different knowledge representation formalisms, including several of the formalisms currently deployed on the Web. Semantically, CL is firmly rooted in FOL[3], whereas its syntax has been designed in a way that accounts for the open spirit of the Web by avoiding syntactic constraints typically occurring in FOL, such as fixed arities of predicates, or the strict distinction between predicate symbols and terms.

The syntactic freedom that CL provides allows for flexible modeling in a higher-order logic (HOL) style and it is even possible to receive some useful HOL-style semantic results, similar to metamodeling in OWL 2 Full. In fact, OWL 2 Full can be translated into CL in an even more natural way than into standard FOL, as demonstrated by Hayes [8]. Furthermore, CL allows to represent the OWL 2 Direct Semantics, as well as other Semantic Web formalisms, such as SWRL and RIF. Compared to all these Semantic Web formalisms, CL is significantly more expressive and flexible and, therefore, users of CL should benefit from extended modeling and reasoning capabilities.

By becoming an ISO standard, CL has taken another dissemination path than the commonly adopted Semantic Web languages. While syntax and semantics of CL are well-defined, software support for editing and managing CL knowledge bases has just started to be developed and support for reasoning in CL is close to non-existent to the best of our knowledge. This arguably constitutes the major obstacle for wide deployment – as we have argued in the preceding section, the often criticized undecidability of CL does not qualify as a sufficient reason to dismiss this formalism right away.

While the development of scalable, ergonomic tools is certainly an endeavor that should not be underestimated, the above presented approach provides a clear strategy for accomplishing readily available state-of-the-art reasoning support. In fact, since the translation of CL into FOL is

[3] Strictly speaking, in order to keep within FOL, one has to avoid the use of so called *sequence markers*.

even more direct than for the two considered variants of OWL, creating a reasoning back-end for CL should be even more straight-forward.

We are convinced that a system capable of reading CL knowledge bases and performing inferences with it would lead to a significant breakthrough toward a wider visibility and practical deployment of CL.

9 Conclusion

In our paper, we have discussed the importance of decidability for practical knowledge representation, putting particular emphasis on well-established Semantic Web languages. On a general level, we argued that from a practical perspective, decidability only provides a qualitative advantage if it comes with a decision algorithm the runtime of which can be guaranteed to be below an acceptable time span. We then identified scenarios where such an algorithm is strictly required as well as scenarios where this is not the case.

We therefore conclude that the necessity to constrain to decidable formalisms strongly depends on the typical automated reasoning tasks to be performed in a knowledge-based system dedicated to a concrete purpose.

In order to still provide necessary reasoning services, we proposed to use available highly optimized implementations of first-order theorem provers and model finders. Realistically, specialized reasoners such as existing OWL 2 DL reasoners are likely to be more efficient on their specific language fragment than generic FOL reasoners. But as these reasoners happen to provide (syntactically restricted) FOL reasoning themselves, all considered reasoners are largely interoperable and, therefore, could be applied in parallel to solve complex reasoning tasks conjointly. This approach offers a smooth transition path towards advanced OWL 2 reasoning without loss of efficiency on existing application areas.

Our own experiments, currently using off-the-shelf standard first-order reasoners, have yielded first encouraging results. We are confident that the good results obtained for undecidable OWL variants will carry over to Semantic Web knowledge representation formalisms that even go beyond the OWL 2 specification, such as SWRL or RIF-BLD combined with OWL 2. We will, in principle, only be limited by what first-order logic provides us.

In the light of these promising results, we strongly believe that the described strategy of providing inferencing services via a translation into FOL and the deployment of first-order reasoning machinery can also pave the way to establishing reasoning support for undecidable formalisms

cherished by the conceptual structures community. Endowing conceptual graphs and common logic with ready-to-use inferencing services along the same lines seems a feasible and worthwhile endeavor and will most likely lead to a more wide-spread adoption of these formalisms among knowledge representation practitioners.

References

1. Andréka, H., van Benthem, J.F.A.K., Németi, I.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic 27(3), 217–274 (1998)
2. Boley, H., Hallmark, G., Kifer, M., Paschke, A., Polleres, A., Reynolds, D. (eds.): RIF Core Dialect. W3C Recommendation (22 June 2010), available at http://www.w3.org/TR/rif-core/
3. Boley, H., Kifer, M. (eds.): RIF Basic Logic Dialect. W3C Recommendation (22 June 2010), available at http://www.w3.org/TR/rif-bld/
4. Brickley, D., Guha, R. (eds.): RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation (10 February 2004), available at http://www.w3.org/TR/rdf-schema/
5. de Bruijn, J. (ed.): RIF RDF and OWL Compatibility. W3C Recommendation (22 June 2010), available at http://www.w3.org/TR/rif-rdf-owl/
6. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. Monatshefte für Mathematik und Physik 38, 173–198 (1931)
7. Gödel, K.: Über die Vollständigkeit des Logikkalküls. Ph.D. thesis, Universität Wien (1929)
8. Hayes, P.: Translating Semantic Web Languages into Common Logic. Tech. rep., IHMC Florida Institute for Human & Machine Cognition, 40 South Alcaniz Street, Pensacola, FL 32502 (18 July 2005), available at http://www.ihmc.us/users/phayes/CL/SW2SCL.html
9. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
10. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B.N., Dean, M.: SWRL: A Semantic Web Rule Language. W3C Member Submission (21 May 2004), available at http://www.w3.org/Submission/SWRL/
11. ISO/IEC JTC 1: Common Logic (CL): A Framework for a Family of Logic-based Languages. No. ISO/IEC 24707:2007(E), ISO International Standard (1 October 2007), available at http://cl.tamu.edu/
12. Kifer, M., Boley, H. (eds.): RIF Overview. W3C Recommendation (22 June 2010), available at http://www.w3.org/TR/rif-overview/
13. Manola, F., Miller, E. (eds.): Resource Description Framework (RDF). Primer. W3C Recommendation (10 February 2004), available at http://www.w3.org/TR/rdf-primer/
14. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language: Profiles. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-profiles/
15. Motik, B., Patel-Schneider, P.F., Cuenca Grau, B. (eds.): OWL 2 Web Ontology Language: Direct Semantics. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-direct-semantics/

16. Motik, B., Patel-Schneider, P.F., Parsia, B. (eds.): OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-syntax/
17. OWL Working Group, W.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-overview/
18. Patel-Schneider, P.F., Motik, B. (eds.): OWL 2 Web Ontology Language: Mapping to RDF Graphs. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-mapping-to-rdf/
19. Rudolph, S., Glimm, B.: Nominals, inverses, counting, and conjunctive queries or: Why infinity is your friend! J. Artif. Intell. Res. (JAIR) 39, 429–481 (2010)
20. Schneider, M. (ed.): OWL 2 Web Ontology Language: RDF-Based Semantics. W3C Recommendation (27 October 2009), available at http://www.w3.org/TR/owl2-rdf-based-semantics/
21. Schneider, M., Sutcliffe, G.: Reasoning in the OWL 2 Full Ontology Language using First-Order Automated Theorem Proving. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Proceedings of the 23rd International Conference on Automated Deduction (CADE 23) (2011), (to appear)
22. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. Journal of Automated Reasoning 43(4), 337–362 (2009)
23. Tsarkov, D., Riazanov, A., Bechhofer, S., Horrocks, I.: Using Vampire to Reason with OWL. In: Proceedings of the Third International Semantic Web Conference (ISWC 2004). LNCS, vol. 3298, pp. 471–485. Springer (2004)
24. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 42(2), 230–265 (1937)
25. Villanueva-Rosales, N., Dumontier, M.: Describing Chemical Functional Groups in OWL-DL for the Classification of Chemical Compounds. In: Golbreich, C., Kalyanpur, A., Parsia, B. (eds.) Proceedings of the 3rd International Workshop on OWL: Experiences and Directions (OWLED 2007). CEUR Workshop Proceedings, vol. 258 (2007), available at http://ceur-ws.org/Vol-258/paper28.pdf
26. Voronkov, A.: Automated Reasoning: Past Story and New Trends. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003). pp. 1607–1612. Morgan Kaufmann Publishers Inc. (2003)

A Reasoning Examples

In this appendix, we will provide examples of reasoning in diverse undecidable languages. There will be one example reasoning problem for each of the undecidable ontology languages that have been discussed earlier in this paper, namely the *OWL 2 direct semantics* when applied to the unrestricted OWL functional syntax (see Sec. 6.1), and the *OWL 2 RDF-based semantics* (Sec. 6.2). The same problems will then also be treated by *Common Logic* reasoning (see Sec. 8).

Each reasoning problem will be given in the form of a knowledge base created from one or more assertions, and there will be two example queries, one being a logically valid conclusion from the knowledge base, and one which does *not* follow. The reasoning examples will first be encoded in the native syntax of the respective language and then be translated into FOL. Diverse FOL reasoners (theorem provers and model-finders) will be applied to each example in order to check (i) whether the knowledge base is semantically consistent, (ii) whether the conclusion query can be inferred, and (iii) whether the non-conclusion query is recognized as a non-entailment. Reasoning results and reasoning durations will be reported for all FOL reasoners. The results will be compared with those of several state-of-the-art OWL 2 DL reasoners in order to check whether the different example problems are really out of scope for existing OWL 2 DL reasoning systems.

We use the *TPTP language* described in [22] as a concrete FOL serialization syntax. The reported reasoning results have been acquired using the *TPTP reasoning service* at `http://tptp.org/cgi-bin/SystemOnTPTP`. The service provides public online access to a large collection of FOL reasoners, and the TPTP language is used as a common input language for all the reasoners.[4] The following *result messages* provided by the service occurred in the experiments:[5]

- `Satisfiable`: semantically consistent set of assertions
- `Unsatisfiable`: semantically inconsistent set of assertions
- `Theorem`: positive entailment
- `CounterSatisfiable`: non-entailment
- `Timeout` (error message): reasoning unfinished within given time limit
- `Error` (error message): reasoner stopped due to a processing error

The following result classification is being applied:

[4] The language and service are maintained by Geoff Sutcliffe, University of Miami.
[5] The full set of result messages of the TPTP reasoning service is defined in `http://tptp.org/cgi-bin/SeeTPTP?Category=Documents&File=SZSOntology`.

– ***correct*** **('+'):** The reasoning process produced a non-error message that matches the expected result.
– ***wrong*** **('−')**: The reasoning process produced a non-error message that does not match the expected result.
– ***unknown*** **('?')**: The reasoning process produced an error message (possibly time-out).

Different kinds of FOL reasoners were used for the different reasoning tasks: FOL theorem provers were used for positive entailment detection, while FOL model-finders were used for knowledge base consistency and non-entailment detection. The following *FOL theorem provers* were used for positive entailment detection:

– E 1.4pre, `http://www.eprover.org`
– iProver 0.9, `http://www.cs.man.ac.uk/~korovink/iprover`
– Prover9 1109a, `http://www.prover9.org`
– SPASS 3.5, `http://www.spass-prover.org`
– Vampire 0.6, `http://www.vprover.org`

The following *FOL model-finders* were used for consistency and non-entailment detection:

– DarwinFM 1.4.5, `http://goedel.cs.uiowa.edu/Darwin`
– iProver-SAT 0.9, `http://www.cs.man.ac.uk/~korovink/iprover`
– Mace4 1109a, `http://www.prover9.org`
– Paradox 4.0, `http://www.cse.chalmers.se/~koen/code/`

In addition, we have applied the following state-of-the-art *OWL 2 DL reasoners*:

– Pellet 2.2.2, `http://clarkparsia.com/pellet`
– HermiT 1.3.2, `http://hermit-reasoner.com`
– FaCT++ 1.5.0, `http://owl.man.ac.uk/factplusplus`

The FOL reasoning experiments were executed with the FOL translations for the different reasoning problems: For *knowledge base consistency* detection experiments, the formulas representing the positive entailment and non-entailment queries have been omitted, and the FOL model-finders have been applied to the remaining formulas. For *positive entailment* detection experiments, the formulas representing the non-entailment query have been omitted, and the FOL theorem provers have been applied to the remaining formulas. Finally, for *non-entailment* detection experiments,

the formulas representing the positive entailment query have been omitted, and the FOL model-finders have been applied to the remaining formulas.[6]

Each reasoning experiment was executed five times. The fastest and the slowest outcomes were ignored as potential outliers, and the average of the remaining three time values was taken the final result duration. A single run included the following steps: starting the reasoning engine, loading the input data, performing reasoning, and printing the output. The measured time for a single run was the "CPU time", i.e., the actual time that the reasoning process was allowed to use the main processors of the computer. The granularity of a measurement was 10ms. The reported results are all 10ms larger than the measured results and have to be understood as the closest possible upper bounds of the real reasoning times. For example, if the "raw" measurement was "0.00s", this means that the reasoning time must have been below 10ms, and so the reported time is "0.01s". For each run, a *time limit of 300 seconds* has been granted to the reasoner.

A.1 Unrestricted OWL 2 Direct Semantics Reasoning

We want to express the notion of a *happy cat couple owner*, being someone who owns a male cat and a female cat which build a couple. Figure 5 shows an example scenario, in which Alice owns the cat couple Max and Fifi, where Max is a male cat and Fifi is a female cat. The relationship between Alice, Max and Fifi has basically a cyclic structure. Our aim is to specify an OWL ontology which enables an OWL reasoner to infer from this scenario that Alice is a happy cat couple owner. It is probably not possible to create an appropriate ontology in OWL 2 DL for this scenario, as OWL 2 DL has been designed in a way that essentially avoids drawing logical conclusions from inherently non-treelike relationships in order to retain decidability of reasoning.

It is, however, possible to give a set of axioms in the OWL 2 functional syntax with the desired formal meaning, as shown below (ontology header and entity declarations are omitted):

[6] *A note on using FOL reasoners in practice:* In all our experiments, the correct reasoning results are known in advance. In practical reasoning applications, where this cannot generally be expected to be the case, it will be advisable to apply theorem provers and model-finders in parallel, and to use the first result message that is not an error message as the result message for the combined system. Only if all FOL reasoners produce error messages, then the result for the combined system would be classified as "unknown".

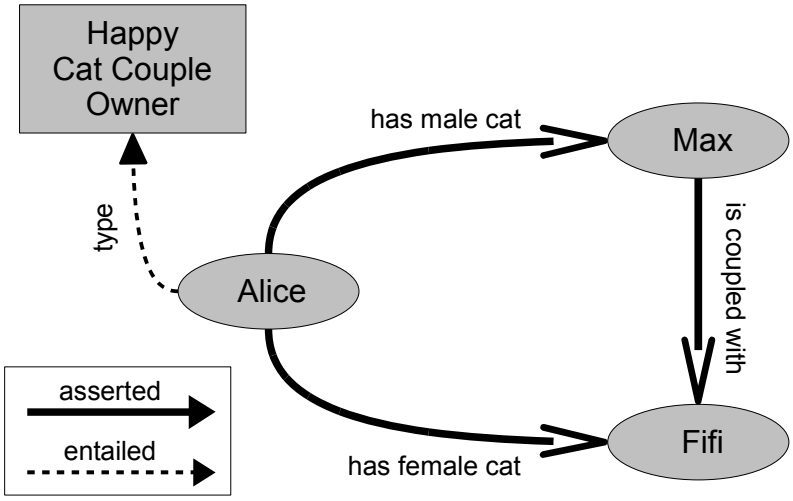

Fig. 5. Example cyclic relationship.

```
SubObjectPropertyOf(
    ObjectPropertyChain(
        ex:hasMaleCat
        ex:isCoupledWith
        ObjectInverseOf(ex:hasFemaleCat) )
    ex:r )

SubClassOf( ObjectHasSelf(ex:r) ex:HappyCatCoupleOwner )

ObjectPropertyAssertion(ex:hasMaleCat ex:alice ex:max)
ObjectPropertyAssertion(ex:hasFemaleCat ex:alice ex:fifi)
ObjectPropertyAssertion(ex:isCoupledWith ex:max ex:fifi)
```

The first axiom is a *sub property chain axiom* on the three properties `ex:hasMaleCat`, `ex:isCoupledWith`, and the inverse of `ex:hasFemaleCat`. The second axiom is a *class subsumption axiom*, which ensures that the individual in entailed to be a `ex:HappyCatCoupleOwner`, whenever there is such a chain of binary relationships built from the three properties, and when the individual at the end of the chain equals the individual at the beginning, i.e., when the chain of relationships builds a cycle. The condition that the two individuals at both ends of the chain are identical is enforced by a *self-restriction* that has been put on the complex property `ex:r` of

the property chain axiom. Doing so hurts one of the global syntactic restriction of OWL 2 DL and, therefore, the above set of axioms is not an OWL 2 DL ontology. Nevertheless, it is still a valid OWL 2 ontology with a precise formal meaning under the OWL 2 direct semantics.

The last three OWL 2 statements above are the encodings of the *property assertions* shown in Fig. 5. As these assertions build a cyclic relationship that matches the conditions of the two axioms above, we expect that `ex:alice` is entailed to be an instance of class `ex:HappyCatCoupleOwner`:

```
ClassAssertion(ex:HappyCatCoupleOwner ex:alice)
```

For the above axioms and assertion data (the knowledge base) the **FOL translation** is now given using the TPTP language. The FOL translation follows the definitions in the specification document of the OWL 2 direct semantics, as discussed in Sec. 6.1. There are *axiom* formulas for the sub property chain axiom and for the class subsumption axiom, as well as *axiom* formulas for the complex expressions used in these axioms: the inverse property expression for the property `ex:hasFemaleCat`, and the self-restriction. The expected positive entailment is encoded as a *conjecture* formula. In addition, for the purpose of non-entailment checking, the following *non-conclusion* statement, which makes the invalid claim that the female cat Fifi is a happy cat couple owner, has been translated into a `conjecture` formula:

```
ClassAssertion(ex:HappyCatCoupleOwner ex:fifi)
```

The complete TPTP serialization is given as follows:

```
%%%%% Knowledgebase: Axioms

fof(kb_axiom_01, axiom, (
    ! [Y0,Y1,Y2,Y3] : (
           ( uri_ex_hasMaleCat(Y0,Y1)
           & uri_ex_isCoupledWith(Y1,Y2)
           & kb_expr_inv(Y2,Y3) )
       => uri_ex_r(Y0,Y3) ))) .
fof(kb_expr_01_01, axiom, (
    ! [X,Y] : ( kb_expr_inv(X,Y) <=> uri_ex_hasFemaleCat(Y,X) ))) .

fof(kb_axiom_02, axiom, (
    ! [X] : ( kb_expr_self(X) => uri_ex_HappyCatCoupleOwner(X) ))) .
fof(kb_expr_02_01, axiom, (
    ! [X] : ( kb_expr_self(X) <=> uri_ex_r(X,X) ))) .

%%%%% Knowledgebase: Data

fof(kb_data_01, axiom, (
```

```
    uri_ex_hasMaleCat(uri_ex_alice, uri_ex_max) )) .
fof(kb_data_02, axiom, (
    uri_ex_hasFemaleCat(uri_ex_alice, uri_ex_fifi) )) .
fof(kb_data_03, axiom, (
    uri_ex_isCoupledWith(uri_ex_max, uri_ex_fifi) )) .

%%%%% Query: Positive Entailment

fof(query_pos, conjecture, (
    uri_ex_HappyCatCoupleOwner(uri_ex_alice) )) .

%%%%% Query: Non-Entailment

fof(query_neg, conjecture, (
    uri_ex_HappyCatCoupleOwner(uri_ex_fifi) )) .
```

Table 1 shows the **results for the FOL reasoners**. Consistency of the knowledge base axioms was confirmed by all model-finders within less than 10ms in average. The positive entailment was detected by all theorem provers, with SPASS needing at most 30ms, and all other reasoners needing less than 10ms. Finally, the non-entailment was detected by all model-finders, with Paradox needing at most 30ms, and all other reasoners needing less than 10ms. In summary, for this problem the use of FOL reasoners turned out to be a safe and efficient choice.

	Reasoner	***Message***	***Time/s***	***Result***
KB Consistency	DarwinFM	Satisfiable	0.01	+
	iProver-SAT	Satisfiable	0.01	+
	Mace4	Satisfiable	0.01	+
	Paradox	Satisfiable	0.01	+
Pos. Entailment	E	Theorem	0.01	+
	iProver	Theorem	0.01	+
	Prover9	Theorem	0.01	+
	SPASS	Theorem	0.03	+
	Vampire	Theorem	0.01	+
Non-Entailment	DarwinFM	CounterSatisfiable	0.01	+
	iProver-SAT	CounterSatisfiable	0.01	+
	Mace4	CounterSatisfiable	0.01	+
	Paradox	CounterSatisfiable	0.03	+

Table 1. Results: OWL 2 direct semantics (unrestricted)

The **OWL 2 DL** reasoners were applied to the OWL 2 functional syntax encoding of the problem. *Pellet* correctly confirmed consistency

of the knowledge base and successfully detected the non-entailment, but wrongly classified the positive entailment example as a non-entailment. In all three cases, Pellet produced a warning message, stating that the sub property chain axiom was ignored while doing reasoning. *Hermit* confirmed knowledge base consistency and classified the non-entailment and the positive entailment as non-entailments. *FaCT++* signaled an error in all three cases ("*non-simple object property ex:r is used as a simple one*"). In summary, all OWL 2 DL reasoners failed to infer the positive entailment.

A.2 OWL 2 RDF-Based Semantics Reasoning

In this example, we will investigate reasoning based on *metamodeling*. Starting from the class of all *persons*, we define the class of *person groups* as the class of all subclasses of the person class. In other words, whenever a class is a subclass of the class of persons (including the person class itself), then this subclass should be entailed to be an *instance* of the class of person groups. In addition, we will introduce a *has-meta-type* property to indicate for any concrete instance of the person class that it has the class of person groups as its *meta-class*. Figure 6 shows the situation for the example class of *happy cat owners*, which is a subclass of all persons, and for the concrete happy cat owner *Alice*. The aim is to be able to infer that the class of happy cat owners is a person group, and that Alice has the class of person groups as its meta-class.

This scenario can be represented in OWL 2 Full, i.e. by means of RDF and the OWL 2 RDF-based semantics. The first three-triple block in the RDF graph given below represents a *has-value restriction* on the OWL vocabulary property `rdfs:subClassOf` and the class of all persons, named `ex:Person`. The restriction represents the set of all subclasses of `ex:Person`. The next triple in the RDF graph represents a *class subsumption* between the has-value restriction and the class of person groups, denoted by URI `ex:PersonGroup`. By this, all sub classes of `ex:Person` become instances of class `ex:PersonGroup`. The following statement in the RDF graph defines the property `ex:hasMetaType` for relating an individual with its meta-class as a *property chain* with a two-times application of the vocabulary property `rdf:type`.

```
_:x rdf:type owl:Restriction ;
    owl:onProperty rdfs:subClassOf ;
    owl:hasValue ex:Person .
```

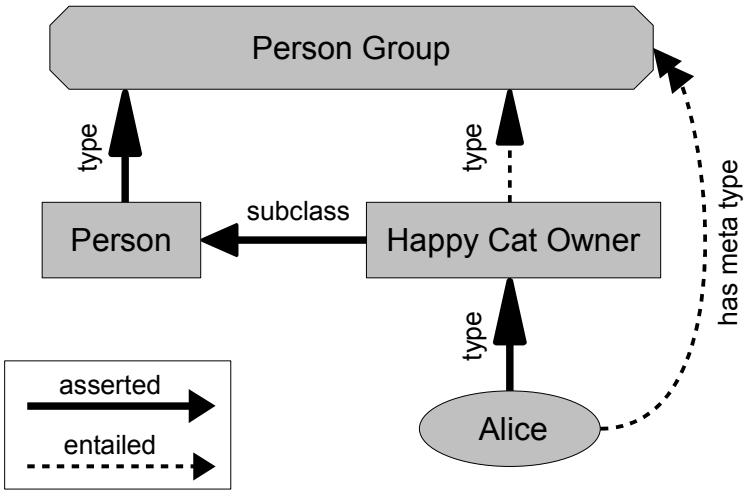

Fig. 6. Example meta-class scenario.

```
_:x rdfs:subClassOf ex:PersonGroup .

ex:hasMetaType owl:propertyChainAxiom ( rdf:type rdf:type ) .

ex:Person rdf:type ex:PersonGroup .
ex:HappyCatOwner rdfs:subClassOf ex:Person .
ex:alice rdf:type ex:HappyCatOwner .
```

The last three RDF triples in the graph are the asserted relationships from Fig. 6. As discussed earlier, we expect to infer that `ex:PersonGroup` has `ex:HappyCatOwner` as an instance and is a meta-class for `ex:alice`, which is expressed by the following query graph:

```
ex:HappyCatOwner rdf:type ex:PersonGroup .
ex:alice ex:hasMetaType ex:PersonGroup .
```

The **FOL translation** of the RDF encoding follows the discussion in Sec. 6.2. The RDF graph is translated into a single formula that consists of a conjunction of ternary atoms, with existentially quantified variables that correspond to the blank nodes in the original graph. In addition, the FOL translation of the OWL 2 RDF-Based semantic conditions is provided. Actually, our translation is restricted to a small subset of the semantic conditions which turns out to be sufficient to prove the positive entailment.

The positive entailment query is encoded as a `conjecture` formula. In addition, for the purpose of non-entailment checking, the following *non-conclusion* statement, which makes the invalid claim that the individual Alice is a direct instance of the class of person groups, has been translated into a `conjecture` formula:

```
ex:alice rdf:type ex:PersonGroup .
```

The complete TPTP serialization is given as follows:

```
%%%%% Knowledgebase: Axioms and Data

fof(kb_axioms_and_data, axiom, (
    ? [B0, B1, B2] : (
          iext(uri_rdf_type, B0, uri_owl_Restriction)
        & iext(uri_owl_onProperty, B0, uri_rdfs_subClassOf)
        & iext(uri_owl_hasValue, B0, uri_ex_Person)
        & iext(uri_rdfs_subClassOf, B0, uri_ex_PersonGroup)
        & iext(uri_owl_propertyChainAxiom, uri_ex_hasMetaType, B1)
        & iext(uri_rdf_first, B1, uri_rdf_type)
        & iext(uri_rdf_rest, B1, B2)
        & iext(uri_rdf_first, B2, uri_rdf_type)
        & iext(uri_rdf_rest, B2, uri_rdf_nil)
        & iext(uri_rdf_type, uri_ex_Person, uri_ex_PersonGroup)
        & iext(uri_rdfs_subClassOf, uri_ex_HappyCatOwner, uri_ex_Person)
        & iext(uri_rdf_type, uri_ex_alice, uri_ex_HappyCatOwner) ))) .

%%%%% Semantic Conditions (Sufficient Subset)

% Has-value restriction
% (Section 5.6 of OWL 2 RDF-Based Semantics)
fof(semcond_owl_restrict_hasvalue, axiom, (
    ! [Z, P, A] : (
            ( iext(uri_owl_hasValue, Z, A)
            & iext(uri_owl_onProperty, Z, P) )
        => (
            ! [X] : (
                    icext(Z, X)
                <=>
                    iext(P, X, A) ))))) .

% Class subsumption
% (Section 5.8 of OWL 2 RDF-Based Semantics)
fof(semcond_owl_rdfsext_subclassof, axiom, (
    ! [C1, C2] : (
            iext(uri_rdfs_subClassOf, C1, C2)
        <=> (
              ic(C1)
            & ic(C2)
            & ( ! [X] : (
                    icext(C1, X)
                =>
                    icext(C2, X) ))))) .
```

```
% Sub property chains
% (Section 5.11 of OWL 2 RDF-Based Semantics)
fof(semcond_owl_chain_002, axiom, (
    ! [P, S1, P1, S2, P2] : (
            ( iext(uri_rdf_first, S1, P1)
            & iext(uri_rdf_rest, S1, S2)
            & iext(uri_rdf_first, S2, P2)
            & iext(uri_rdf_rest, S2, uri_rdf_nil) )
        => (
               iext(uri_owl_propertyChainAxiom, P, S1)
           <=> (
                 ip(P)
               & ip(P1)
               & ip(P2)
               & ( ! [Y0, Y1, Y2] : (
                        ( iext(P1, Y0, Y1)
                        & iext(P2, Y1, Y2) )
                    =>
                        iext(P, Y0, Y2) ))))))) .

% Definition of part "IP"
% (Section 3.1 of RDF Semantics)
fof(semcond_rdf_type_ip, axiom, (
    ! [P] : (
            iext(uri_rdf_type, P, uri_rdf_Property)
        <=>
            ip(P) ))) .

% Definition of mapping "ICEXT"
% (Section 4.1 of RDF Semantics)
fof(semcond_rdfs_cext_def, axiom, (
    ! [X, C] : (
            iext(uri_rdf_type, X, C)
        <=>
            icext(C, X) ))) .

% Definition of part "IC"
% (Section 4.1 of RDF Semantics)
fof(semcond_rdfs_ic_def, axiom, (
    ! [X] : (
            ic(X)
        <=>
            icext(uri_rdfs_Class, X) ))) .

% Property extension of "owl:onProperty"
% (Section 5.3 of OWL 2 RDF-Based Semantics)
fof(semcond_owl_prop_onproperty_ext, axiom, (
    ! [X, Y] : (
            iext(uri_owl_onProperty, X, Y)
        => (
             icext(uri_owl_Restriction, X)
           & ip(Y) )))) .

% Property extension of "owl:hasValue"
% (Section 5.3 of OWL 2 RDF-Based Semantics)
fof(semcond_owl_prop_hasvalue_ext, axiom, (
    ! [X, Y] : (
```

```
            iext(uri_owl_hasValue, X, Y)
        => (
              icext(uri_owl_Restriction, X)
            & ir(Y) )))) .

%%%%% Query: Positive Entailments

fof(query_pos, conjecture, (
      iext(uri_rdf_type, uri_ex_HappyCatOwner, uri_ex_PersonGroup)
    & iext(uri_ex_hasMetaType, uri_ex_alice, uri_ex_PersonGroup) )) .

%%%%% Query: Non-Entailments

fof(query_neg, conjecture, (
    iext(uri_rdf_type, uri_ex_alice, uri_ex_PersonGroup) )) .
```

Table 2 shows the **results for the FOL reasoners**. Knowledge base consistency was confirmed by all model-finders. *iProver-SAT* took about half a second per run, while *Paradox* needed less than 1/10s in average, and *DarwinFM* and *Mace4* even finished in less than 10ms. All theorem provers detected the positive entailment. *Vampire* required most time by using ca. 8.5s, while the reasoning times for all other reasoners remained clearly under 1s in average, the fastest reasoner being *iProver*, which always stopped after less than 20ms. Finally, the non-entailment was detected by all but one model-finder: *Mace4* always timed out, which means that it would have required more than 300s to finish the task. DarwinFM terminated after more than one minute. The two other reasoners always found a solution in about 0.2s. In summary, for this problem the use of FOL reasoners turned out to be safe (there were no wrong results) and in many cases efficient, although there were some runs that were slow up to time out.

It needs to be mentioned that the positive results for knowledge base consistency and non-entailment detection have to be treated with caution. All reasoning experiments were executed on a translation of only a small subset of the OWL 2 RDF-Based semantic conditions, which was chosen to be sufficient for proving the positive entailment. Since the complete set of premise FOL axioms is incomplete, it is therefore, in theory, possible that the reasoners would produce different results on a complete premise set. More realistically, one could imagine that the reasoners might have more difficulties to produce a result in reasonable time on the much larger set of formulas, when all semantic conditions were taken into account. We will not further investigate this topic here, but point to related experiments executed in [21], where this was really found to be the case.

	Reasoner	*Message*	*Time/s*	*Result*
KB Consistency	DarwinFM	Satisfiable	0.01	+
	iProver-SAT	Satisfiable	0.55	+
	Mace4	Satisfiable	0.01	+
	Paradox	Satisfiable	0.09	+
Pos. Entailment	E	Theorem	0.08	+
	iProver	Theorem	0.02	+
	Prover9	Theorem	0.17	+
	SPASS	Theorem	0.13	+
	Vampire	Theorem	8.52	+
Non-Entailment	DarwinFM	CounterSatisfiable	77.63	+
	iProver-SAT	CounterSatisfiable	0.19	+
	Mace4	Timeout	>300.00	?
	Paradox	CounterSatisfiable	0.23	+

Table 2. Results: OWL 2 RDF-based semantics

The **OWL 2 DL reasoners** were applied to the RDF encoding of the problem. *Pellet* confirmed consistency of the knowledge base and non-entailment for the non-entailment query, but wrongly classified the positive entailment query as a non-entailment. Actually, Pellet listed both component assertions of the positive entailment query as non-entailments separately. *Hermit* confirmed knowledge base consistency and classified the non-entailment and positive entailment query as non-entailments. *FaCT++* correctly confirmed consistency for the knowledge base and non-entailment for the non-entailment query, but signaled an error for the positive entailment query ("*individual name expected in the isInstance()*"). In summary, all OWL 2 DL reasoners failed to infer the positive entailment.

A.3 Common Logic Reasoning and Cyclic Relationships

As it was possible to translate the previously discussed reasoning problems from their OWL-based encoding into FOL, it is obvious that they could be translated into CL in basically the same way. However, as CL is more expressive and flexible than the discussed OWL 2 variants, there may be other, potentially more effective ways to encode the reasoning problems in CL. Further, being a variant of first-order logic, it should be particularly easy to make the CL encodings of the problem available to FOL reasoning. This will be the topic of this and the following section. We will use *CG display form* [11, Annex B] for a graphical presentation of the problems in CL, as well as the two normative CL dialects *CGIF* [11, Annex B] and *CLIF* [11, Annex A] to give the actual CL encoding.

The cyclic relationship problem from Sec. A.1 was basically characterized the following way: *A happy cat couple owner is someone who owns a male cat and a female cat which build a couple.* Under the OWL 2 direct semantics, we had to use a somewhat complicated and non-obvious encoding for this relationship. In CL, however, the relationship can be represented in a straight-forward way. This is shown in Fig. 7, which uses CG display form to visualize the main semantic relationship.

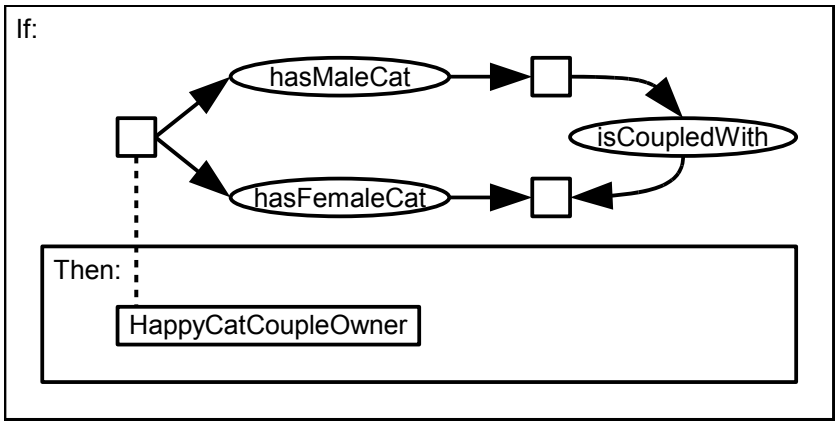

Fig. 7. Cyclic relationship problem in CL (CG display form).

The corresponding encoding in CGIF, now also covering the remaining relationships from the knowledge base, is given as follows:

```
[: @every *x] [: @every *m] [: @every *f]
[If: ("ex:hasMaleCat" ?x ?m)
     ("ex:hasFemaleCat" ?x ?f)
     ("ex:isCoupledWith" ?m ?f)
  [Then: ["ex:HappyCatCoupleOwner": ?x] ]]

("ex:hasMaleCat" "ex:alice" "ex:max")
("ex:hasFemaleCat" "ex:alice" "ex:fifi")
("ex:isCoupledWith" "ex:max" "ex:fifi")
```

Alternatively, the following is the encoding in CLIF:

```
(forall (x m f)
    (if (and (ex:hasMaleCat x m)
             (ex:hasFemaleCat x f)
```

```
            (ex:isCoupledWith m f))
        (ex:HappyCatCoupleOwner x)))

(ex:hasMaleCat ex:alice ex:max)
(ex:hasFemaleCat ex:alice ex:fifi)
(ex:isCoupledWith ex:max ex:fifi)
```

The positive entailment query, which was represented as a class assertion under the OWL 2 functional syntax in Sec. A.1, is encoded as a unary relationship in CL. In CGIF, we receive:

```
["ex:HappyCatCoupleOwner": "ex:alice"]
```

and in CLIF, we get:

```
(ex:HappyCatCoupleOwner ex:alice)
```

Likewise, the non-entailment query is encoded in CGIF as:

```
["ex:HappyCatCoupleOwner": "ex:fifi"]
```

and in CLIF as:

```
(ex:HappyCatCoupleOwner ex:fifi)
```

The CL encoding of the problem does not really make use of the enhanced syntactic flexibility of CL and so the translation into FOL is essentially a 1:1 correspondence. The complete TPTP serialization is given as follows.

```
%%%%% Knowledge Base: Axioms

fof(kb_axiom_01, axiom, (
    ! [X,M,F] : (
            ( uri_ex_hasMaleCat(X,M)
            & uri_ex_hasFemaleCat(X,F)
            & uri_ex_isCoupledWith(M,F) )
        => uri_ex_HappyCatCoupleOwner(X) ))) .

%%%%% Knowledgebase: Data

fof(kb_data_01, axiom, (
    uri_ex_hasMaleCat(uri_ex_alice, uri_ex_max) )) .
fof(kb_data_02, axiom, (
    uri_ex_hasFemaleCat(uri_ex_alice, uri_ex_fifi) )) .
fof(kb_data_03, axiom, (
    uri_ex_isCoupledWith(uri_ex_max, uri_ex_fifi) )) .

%%%%% Query: Positive Entailment

fof(query_pos, conjecture, (
    uri_ex_HappyCatCoupleOwner(uri_ex_alice) )) .
```

```
%%%%% Query: Non-Entailment

fof(query_neg, conjecture, (
    uri_ex_HappyCatCoupleOwner(uri_ex_fifi) )) .
```

Table 3 shows the **results for the FOL reasoners**. All reasoning tasks were successfully solved by all reasoners. The theorem prover *SPASS* needed at most 20ms in average to detect the positive entailment, while all other tasks were completed by all reasoners in less than 10ms. These resulting time values are in line with those for OWL 2 direct semantics reasoning, which is noteworthy, as CL is a much more expressive entailment regime. No results are reported for the OWL 2 DL reasoners, since the scenario discussed here is the same for them as in Sec. A.1.

	Reasoner	***Message***	***Time/s***	***Result***
KB Consistency	DarwinFM	Satisfiable	0.01	+
	iProver-SAT	Satisfiable	0.01	+
	Mace4	Satisfiable	0.01	+
	Paradox	Satisfiable	0.01	+
Pos. Entailment	E	Theorem	0.01	+
	iProver	Theorem	0.01	+
	Prover9	Theorem	0.01	+
	SPASS	Theorem	0.03	+
	Vampire	Theorem	0.01	+
Non-Entailment	DarwinFM	CounterSatisfiable	0.01	+
	iProver-SAT	CounterSatisfiable	0.01	+
	Mace4	CounterSatisfiable	0.01	+
	Paradox	CounterSatisfiable	0.01	+

Table 3. Results: Cyclic-Relationship Experiment in CL

A.4 Common Logic Reasoning and Metamodeling

In Sec. A.2, it was shown how to use metamodeling-based reasoning under the OWL 2 RDF-based semantics, i.e., in OWL 2 Full. As has been mentioned in Sec. 8, CL also enables a form of higher-order-style modeling and to draw conclusions from it. In fact, it is possible to encode the reasoning problem of Sec. A.2 in CL as well. This is shown in Fig. 8. The conceptual graph in the upper half of the figure represents the relationship

from Sec. A.2 that *whenever a class is a subclass of the class of persons, then this subclass is an instance of the class of person groups.* The lower half of the figure represents the relationship that *for an instance of any subclass of the person class, the person has the class of person groups as its meta-class.*

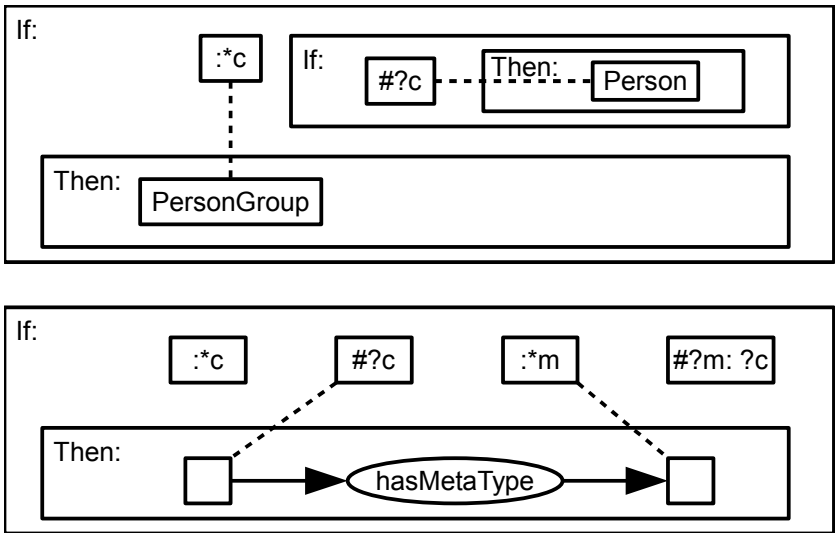

Fig. 8. Metamodeling problem in CL (CG display form).

In CGIF, the full encoding is given as:

```
[: @every *c]
[If: [: @every *x]
     [If: [#?c: ?x] [Then: ["ex:Person": ?x] ]]
  [Then: ["ex:PersonGroup": ?c] ]]

[: @every *x] [: @every *m]
[If: [: *c] [#?c: ?x] [#?m: ?c]
  [Then: ("ex:hasMetaType" ?x ?m) ]]

["ex:PersonGroup": "ex:Person"]

[: @every *x]
[If: ["ex:HappyCatOwner": ?x]
  [Then: ["ex:Person": ?x] ]]
```

```
["ex:HappyCatOwner": "ex:alice"]
```

In CLIF, we receive:

```
(forall (c)
    (if (forall (x) (if (c x) (ex:Person x)))
        (ex:PersonGroup c)))

(forall (x m)
    (if (exists (c) (and (c x) (m c)))
        (ex:hasMetaType x m)))

(ex:PersonGroup ex:Person)

(forall (x)
    (if (ex:HappyCatOwner x) (ex:Person x)))

(ex:HappyCatOwner ex:alice)
```

The positive entailment query, as originally given in Sec. A.2, has the following form in CGIF:

```
["ex:PersonGroup": "ex:HappyCatOwner"]
("ex:hasMetaType" "ex:alice" "ex:PersonGroup")
```

In CLIF, the representation is as follows:

```
(and (ex:PersonGroup ex:HappyCatOwner)
     (ex:hasMetaType ex:alice ex:PersonGroup))
```

Finally, the encoding of the non-entailment query is, in CGIF:

```
["ex:PersonGroup": "ex:alice"]
```

and in CLIF:

```
(ex:PersonGroup ex:alice)
```

A direct **FOL translation** of the CL encoding is not possible due to the fact that some of the names that occur in the CL formulas are used both as predicate names and constant names and since there is quantification over variables that are used in predicate position. To cope with this situation, we will translate any n-ary atom $p(x_1, ..., x_n)$ into a corresponding $(n+1)$-ary atom $\text{cl\_apply}_{n+1}(p, x_1, ..., x_n)$, i.e., the predicate name "p"

is replaced by a constant term of the same name, and the extension of the n-ary predicate p is embedded in the extension of the $(n+1)$-ary predicate cl_apply$_{n+1}$ with a fixed first component. This approach is well-justified, as CL allows to treat predicates in a "*non-segregated*" way, where the denotations of all predicate names are members of the universe of discourse UD_I, while the semantics of CL ensures that the predicate names still represent subsets of UD_I^n; see [11] for further information. The basic approach used here for translating CL into standard FOL has already been used in the past, for example, in the *Extended Common Logic (ECL)* draft.[7] The complete TPTP serialization is given as follows:

```
%%%%% Knowledge Base: Axioms

fof(kb_axiom_01, axiom, (
    ! [C] : ( (! [X] : (cl_apply_2(C, X) => cl_apply_2(uri_ex_Person, X) ))
        => cl_apply_2(uri_ex_PersonGroup, C) ))) .

fof(kb_axiom_02, axiom, (
    ! [X, M] : ( (? [C] : (cl_apply_2(C, X) & cl_apply_2(M, C) ))
        => cl_apply_3(uri_ex_hasMetaType, X, M) ))) .

%%%%% Knowledge Base: Data

fof(kb_data_01, axiom, (
    cl_apply_2(uri_ex_PersonGroup, uri_ex_Person) )) .
fof(kb_data_02, axiom, (
    ! [X] : ( cl_apply_2(uri_ex_HappyCatOwner, X)
        => cl_apply_2(uri_ex_Person, X) ))) .
fof(kb_data_03, axiom, (
    cl_apply_2(uri_ex_HappyCatOwner, uri_ex_alice) )) .

%%%%% Query: Positive Entailment

fof(query_pos, conjecture, (
      cl_apply_2(uri_ex_PersonGroup, uri_ex_HappyCatOwner)
    & cl_apply_3(uri_ex_hasMetaType, uri_ex_alice, uri_ex_PersonGroup) )) .

%%%%% Query: Non-Entailment

fof(query_neg, conjecture, (
    cl_apply_2(uri_ex_PersonGroup, uri_ex_alice) )) .
```

Table 4 shows the **results for the FOL reasoners**. Knowlegebase consistency and the non-entailment was confirmed by all model-finders in at most 10ms in average. The positive entailment was detected by all theorem provers, the slowest being SPASS taking at most 20ms, all others

[7] ECL 0.4 (Tammet, 2004): `http://cs.ttu.ee/kursused/wav4130/Elm/ecl.html`

taking less than 10ms. This result is considerably better than for reasoning under the OWL 2 RDF-based semantics, where processing often took much longer up to time out in some cases. It is also noteworthy that, compared to the experiments for the OWL 2 RDF-based semantics, the results for consistency and non-entailment checking are perfectly reliable, since the problem encoding for the case of CL reasoning is complete. It is an interesting observation that switching to a more expressive formalism, such as CL, can sometimes lead to significantly higher reasoning performance. No results are reported for the OWL 2 DL reasoners, since the scenario discussed here is the same for them as in Sec. A.2.

	Reasoner	***Message***	***Time/s***	***Result***
KB Consistency	DarwinFM	Satisfiable	0.01	+
	iProver-SAT	Satisfiable	0.01	+
	Mace4	Satisfiable	0.01	+
	Paradox	Satisfiable	0.01	+
Pos. Entailment	E	Theorem	0.01	+
	iProver	Theorem	0.01	+
	Prover9	Theorem	0.01	+
	SPASS	Theorem	0.03	+
	Vampire	Theorem	0.01	+
Non-Entailment	DarwinFM	CounterSatisfiable	0.01	+
	iProver-SAT	CounterSatisfiable	0.01	+
	Mace4	CounterSatisfiable	0.01	+
	Paradox	CounterSatisfiable	0.01	+

Table 4. Results: Metamodeling Experiment in CL