Satisfiability Model Visualization Plugin for Deep Consistency Checking of OWL Ontologies

Martins Barinskis and Guntis Barzdins

Institute of Mathematics and Computer Science
University of Latvia
martins.barinskis@gmail.com, guntis.barzdins@mii.lu.lv

Abstract. We present an original *Protégé* plugin developed for the deep consistency checking of OWL ontologies. The plugin constructs and visualizes a minimal satisfiability model of the ontology, which is likely to uncover potential ontological errors: if the constructed model contradicts the author's intentions, then the ontology itself is either wrong or incomplete. A satisfiability model is generated using *Mace4*, a first-order logic (FOL) finite model builder, from the FOL formulas corresponding to the OWL ontology definition. The constructed satisfiability model is visualized using an original music score notation plugin of *Protégé*.

1 Introduction

The ontology satisfiability is a property that indicates whether all classes defined in the ontology are satisfiable. A class is deemed to be unsatisfiable if it cannot possibly have any instances [1]. Satisfiability implies consistency.

The problem of checking consistency (i.e., finding the existence of a model) of an arbitrary FOL formula is not decidable. However, for the description logic fragments of FOL, on which OWL DL (SHOIN) and OWL 1.1 (SROIQ) are based, satisfiability (consistency) checking is decidable [2]. The only "inconvenience" with these description logic fragments is that in some cases their only satisfiability (consistency) model can turn out to be infinite [3]. This is not a problem for the tableau algorithm at the heart of FaCT++[4] and Pellet[5] reasoners, which wind this infinite model into a finite structure. However, a consequence of this "inconvenience" is that FaCT++ and Pellet reasoners do not generate the actual satisfiability (consistency) model.

Meanwhile it would be problematic to use a consistent ontology with only infinite satisfiability model in the Semantic Web context - no finite set of individuals (raw RDF data) would ever satisfy conditions of such ontology. Therefore further we consider only ontologies with finite satisfiability models.

The *Protégé* plugin described in this paper [6] uses *Mace4* [7], a generic FOL finite model builder, to generate and visualize a minimal finite satisfiability model of an ontology. The proposed approach complements the traditional ontology debugging tools[8]: if the automatically constructed minimal model contradicts the author's intentions, the ontology itself is either wrong or incomplete.

2 Ontology Description Using First-Order Logic

To illustrate our approach, let us consider a simple pizza ontology mapped to first-order logic predicates, both shown side-by-side:

Ontology(	formulas(sos).

Class(Pizza partial
  restriction(hasTopping
  someValuesFrom(PizzaTopping))
  owl:Thing)
	(all a Pizza(a)-> (exists b PizzaTopping(b) & hasTopping(a,b)))

Class(MeatyPizza partial
  restriction(hasTopping
  someValuesFrom(MeatTopping))
  Pizza)
	& (all x MeatyPizza(x)->Pizza(x))
	& (all a MeatyPizza(a) -> (exists b MeatTopping(b) & hasTopping(a,b)))

Class(CheeseOnlyPizza partial
  restriction(hasTopping
  someValuesFrom(CheeseTopping))
  Pizza
  restriction(hasTopping
  allValuesFrom(CheeseTopping)))
	& (all x CheeseOnlyPizza(x)->Pizza(x))
	& (all a CheeseOnlyPizza(a)->(exists b CheeseTopping(b) & hasTopping(a,b)))
	& (all a all b CheeseOnlyPizza(a) & hasTopping(a,b)->CheeseTopping(b))

Class(PizzaTopping partial owl:Thing)	
Class(CheeseTopping partial PizzaTopping)	& (all x CheeseTopping(x) ->PizzaTopping(x))
Class(MeatTopping partial PizzaTopping)	& (all x MeatTopping(x) ->PizzaTopping(x))
DisjointClasses(CheeseTopping MeatTopping)	& (all a all b MeatTopping(a) & CheeseTopping(b)->-(a=b))
DisjointClasses(PizzaTopping Pizza)	& (all a all b PizzaTopping(a) & Pizza(b)->-(a=b))
ObjectProperty(hasTopping domain(Pizza) range(PizzaTopping))	& (all a all b hasTopping(a,b) -> Pizza(a) & PizzaTopping(b))
	& (exists x1 exists x2 exists x3 exists x4 exists x5 exists x6 MeatyPizza(x1) & CheeseOnlyPizza(x2) & CheeseTopping(x3) & MeatTopping(x4) & Pizza(x5) & PizzaTopping(x6)).
)	end_of_list.

The satisfiability (which is a stronger requirement than consistency) is asserted in the last conjunction member of our FOL formula, i.e. we require that there exists an individual in every class declared in the ontology.

The conversion from OWL to first-order logic is based on simple pattern matching as illustrated by the example above.

3 Acquiring Satisfiability Model

The FOL syntax shown in the above listing is accepted by *Mace4*, a FOL model builder. It constructs a minimal (in the number of individuals) satisfiability model for our ontology.

For the FOL formula shown above as input, *Mace4* yields the following output (irrelevant lines have been stripped out):

```
interpretation( 4, [number=1, seconds=0], [
        relation(CheeseOnlyPizza(_), [ 0, 0, 0, 1 ]),
        relation(CheeseTopping(_), [ 0, 1, 0, 0 ]),
        relation(MeatTopping(_), [ 0, 0, 1, 0 ]),
        relation(MeatyPizza(_), [ 1, 0, 0, 0 ]),
        relation(Pizza(_), [ 1, 0, 0, 1 ]),
        relation(PizzaTopping(_), [ 0, 1, 1, 0 ]),
        relation(hasTopping(_,_), [
                          0, 1, 1, 0,
                          0, 0, 0, 0,
                          0, 0, 0, 0,
                          0, 1, 0, 0 ])
]).
```

This result shows that the minimal model is of size 4, i.e., the satisfiability model of the given ontology consists of 4 individuals, a_1, a_2, a_3 and a_4. From *Mace4* output one can see that, for example, individual a_2 belongs to classes *CheeseTopping* and *PizzaTopping*, while relation *hasTopping* is established between individual pairs $a_1 - a_2$, $a_1 - a_3$ and $a_4 - a_2$.

4 Satisfiability Model Visualization

The satisfiability model created with *Mace4* can be visualized using an original "music score notation" shown in fig. 1, which in our opinion is rather intuitive for the debugging purposes. In this notation classes (gray) and individuals (green) are visualized as lines that are interconnected by the notes representing the predicates (relations) between the individuals (red) or between individuals and classes (blue). The roles (domain, range) of the predicate (relation) arguments could be written on the legs of the notes (useful for n-ary predicates), but is omitted in fig. 1 for the sake of intelligibility.

The visualization is implemented as a plugin for the popular ontology editor *Protégé*. It forms the FOL formula from the OWL ontology currently being edited, passes it to the *Mace4* model builder, retrieves the satisfiability model and finally visualizes it.

5 Conclusions and Further Work

The proposed approach complements the traditional ontology debugging tools with the possibility to identify ontologies that have trivial models (under-constrained ontologies). With the introduction of more feature-rich ontology languages, such as OWL 1.1 or even larger subsets of FOL [9], the usefulness of such automated debugging tools is likely to grow.

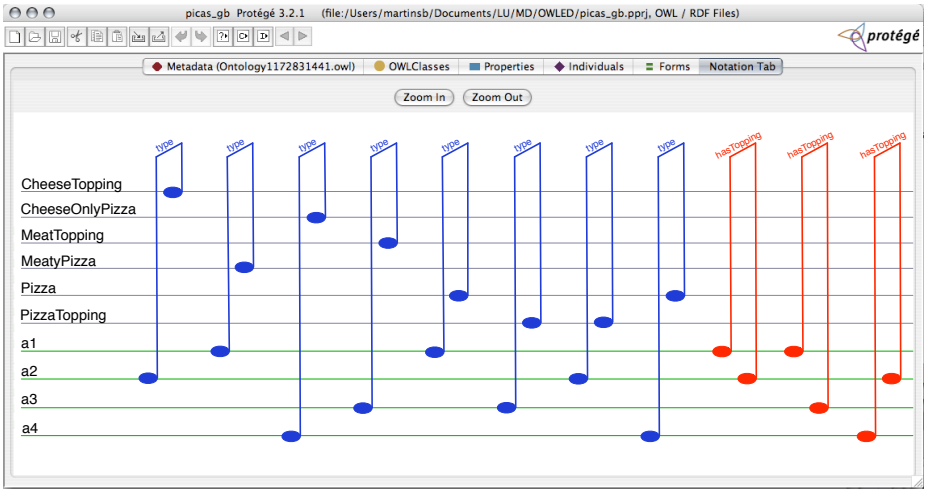

Fig. 1. Example pizza ontology minimal satisfiability model visualization

An obvious limitation of the proposed approach is the scalability in terms of the model size both for *Mace4* model builder and for the "music score notation" to still be useful. The *Mace4* scaling issue could be addressed by extending the native FaCT++ or Pellet reasoners with the model generation functionality (for cases when the model is determined to be finite). The "music score notation" visualization could be complemented with filtering options to limit the amount of concurrently displayed information.

References

1. Horrocks, I., Patel-Schneider, P.F.: Reducing owl entailment to description logic satisfiability. LNCS (2870) (2003) 17–29
2. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible sroiq. In: KR 2006, AAAI Press (2006) 57–67
3. Calvanese, D.: Finite model reasoning in description logics. In: Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR'96), Morgan Kaufmann, Los Altos (1996)
4. OWL: Fact++. http://owl.man.ac.uk/factplusplus/
5. Pellet: An open source owl-dl reasoner in java. http://pellet.owldl.com/
6. Barinskis, M., Barzdins, G.: Satisfiability model visualizer for protégé. http://apps.lumii.lv/satmodviz/index.html
7. McCune, W.: Mace4 reference manual and guide. Technical Report 264, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439 (August 2003)
8. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging owl ontologies. In: WWW'05, ACM Press (2005) 633–640
9. Horrocks, I., Voronkov, A.: Reasoning support for expressive ontology languages using a theorem prover. In: FoIKS. Number 3861 in LNCS (2006) 201–218