# Consistency Checking of Semantic Web Ontologies

Kenneth Baclawski[1], Mieczyslaw M. Kokar[2], Richard Waldinger[4], and Paul A. Kogut[3]

[1] College of Computer Science, Northeastern University
[2] Department of Electrical and Computer Engineering, Northeastern University
[3] Lockheed Martin Management and Data Systems
[4] SRI International

**Abstract.** Ensuring that ontologies are consistent is an important part of ontology development and testing. This is especially important when autonomous software agents are to use ontologies in their reasoning. Reasoning with inconsistent ontologies may lead to erroneous conclusions. In this paper we introduce the ConsVISor tool for consistency checking of ontologies. This tool is a consistency checker for formal ontologies, including both traditional data modeling languages and the more recent ontology languages. ConsVISor checks consistency by verifying axioms. ConsVISor is part of the UBOT toolkit that uses a variety of techniques such as theorem proving and logic programming. Some examples of the use of these tools are given.

## 1 Introduction to ConsVISor

Formal ontologies are fundamental for the Semantic Web. They are especially important for autonomous software agents for which a shared ontology is necessary for meaningful communication. However, because autonomous software agents perform their reasoning and come to conclusions without human supervision, it is essential that the shared ontology be consistent. If an ontology is inconsistent, then any conclusion may be deduced.

The ConsVISor tool is a consistency checker for formal ontologies. ConsVISor can check consistency for a variety of languages. It currently supports class diagrams specified in the Unified Modeling Language (UML) [1], and formal ontologies specified in RDF [11] and DAML+OIL [3]. ConsVISor is part of the UBOT toolkit [14]. In addition to UML, RDF and DAML+OIL, the UBOT toolkit can be used to check the consistency of logical theories specified using the Knowledge Interchange Format (KIF) [5].

The architecture of ConsVISor is shown in Figure 1. The ontology file is first translated from the input ontology language to the language required by a logic programming engine. The translation step incorporates some of the semantics of DAML. For example, if two names are explicitly stated to represent equivalent resources (by using a property such as `daml:equivalentTo` or `daml:sameClassAs`,

I. Horrocks and J. Hendler (Eds.): ISWC 2002, LNCS 2342, pp. 454–459, 2002.

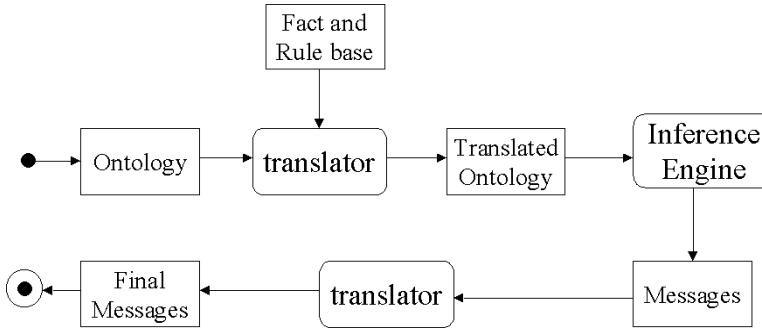


**Fig. 1.** ConsVISor Architecture

then the names are mapped to the same internal name. A background framework (fact and rule base) is combined with the translated ontology and also used by the logic programming engine. The output of the logic programming engine is translated back to a form compatible with the input ontology language and presented to the user.

The ConsVISor tool currently uses Prolog as its logic programming engine, but we are in the process of adding support for other engines such as Jess [7]. ConsVISor not only checks all of the axioms of the ontology, but it also checks for situations that might be mistakes even though they are not inconsistencies. Such cases are less severe than inconsistencies and can be suppressed if requested. For example, suppose that an ontology developer inadvertently misspelled a class name at one place in an ontology. This is both syntactically correct and semantically consistent because one can infer that the misspelled name is a class by the fact that it has been used as a class. Yet it is clearly a mistake, and finding such situations is of great practical benefit. Another example of a typical mistake in ontology development is given in Section 3 below.

If no error messages or warnings are printed by ConsVISor, then the ontology is consistent. However, an ontology might be consistent even though ConsVISor prints warnings. For example, ConsVISor does not support paramodulation. In particular, this means that ConsVISor is implicitly assuming that if two resources have different names and are not explicitly specified to be the same, then they are distinct resources. In other words, if $s$ and $t$ are resources that are mapped to distinct names by the translation step (see Figure 1), then ConsVISor adds the axiom $not(s = t)$. More generally, ConsVISor uses "negation as failure" rather than logical negation. The limitations of ConsVISor are a consequence of its use of Prolog, whose logical limitations are well known. In addition, ConsVISor cannot check the consistency of the logical system within which it resides (i.e., KIF, RDF, RDFS and DAML+OIL). To deal with these limitations, one can make use of another component of UBOT. Section 4 gives an example of the use of the SNARK theorem prover to find a logical inconsistency in the axioms for RDF.

If there is an inconsistency, then there is a good chance that SNARK can find it. Unlike the error messages and warnings produced by ConsVISor, these are true inconsistencies, not just possible mistakes. Since SNARK need not terminate, it either finds an inconsistency or it gives up when it runs out of time or space.

In Section 5 we conclude the paper and discuss some of the future work we have planned for ConsVISor and UBOT.

## 2 Related Work

There are a number of other systems for consistency checking. The OilEd [10] ontology editor that is intended for small-scale ontology development and consistency checking. It is not a complete ontology development environment. OilEd uses FaCT for its consistency checking. FaCT [6] is description logic classifier that can also be used for modal logic satisfiability testing. It can check the consistency of a DAML+OIL ontology, but it cannot check DAML+OIL itself. Furthermore, one can use FaCT only if one imposes some additional limitations on a DAML+OIL ontology that go beyond those of DAML+OIL itself. For example, one cannot have a cardinality restriction on a transitive property. Finally, FaCT only checks consistency, it does not issue warnings that indicate possible mistakes that are not inconsistencies in themselves.

JTP is a theorem prover written in Java [4]. JTP accepts KIF axioms, but it doesn't support paramodulation and only accepts axioms in Horn clause form. Since the DAML+OIL axioms contain equalities, equivalences and other non-Horn structures, it is not compatible with DAML+OIL.

Chimæra is a software system that supports users in creating and maintaining distributed ontologies on the web [2]. Two major functions it supports are merging multiple ontologies together and diagnosing individual or multiple ontologies. It supports users in such tasks as loading knowledge bases in differing formats, reorganizing taxonomies, resolving name conflicts, browsing ontologies, editing terms, etc. While the Chimæra system is an effective tool for ontology integration, its diagnostic suite is currently limited and not connected to a full theorem prover [9].

## 3 The Expression/Operation Ontology

In this section we give some an example of an inconsistent ontology that can arise in ontology development, and the results of running ConsVISor on it. The example is an ontology for expressions consisting of binary or higher operators that can be combined recursively. For example, $(x+y+5)*(z+3)*(a+b)$ would be such an expression. This includes operators such as addition and multiplication. In the first diagram, the notion of Expression is introduced with two exclusive subtypes, Elementary and Operation. An elementary expression has no further substructure. It would include constants and variables. This ontology is shown in Figure 2 using UML. An association, called *operands* between Expression and

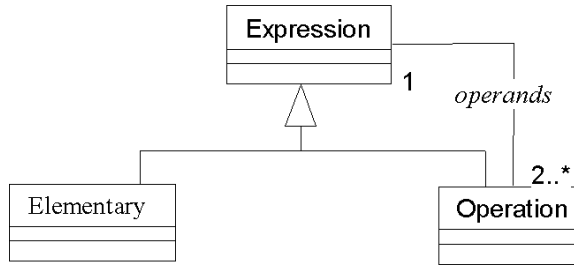


**Fig. 2.** Expression/Operation Ontology

Operation specifies that an operation is a subexpression that combines operands that are either elementary expressions or other operations.

The Expression/Operation ontology is not obviously inconsistent. The cardinality restrictions of the association between Operation and Expression specify that there are at least twice as many instances of Operation as there are instances of Expression. However, Operation is a subtype of Expression, so every instance of Operation is also an instance of Expression. Using the symbol `#` to mean *the number of instances*, we have shown that:

$$\#\text{Expression} \geq \#\text{Operation} \geq 2\#\text{Expression}$$

which implies that the Operation class (as well as the Expression class) is either empty or has an infinite number of elements.

The problem with this ontology is that the cardinality restrictions are in the wrong order. Reversing cardinality restrictions is a common mistake in data modeling languages. The ConsVISor tool will, in this case, warn the user that some of the classes cannot be instantiated.

## 4 Functional Property Example

The example in this section considers a logical theory that cannot be checked by the ConsVISor approach, and it illustrates the role of theorem proving in consistency checking. The theorem prover in the UBOT toolkit is SNARK [12,13]. The principal inference rules used by SNARK are resolution and paramodulation. Some distinctive features of SNARK are its support for special unification algorithms, sorts, nonclausal formulas, answer construction for program synthesis, procedural attachment, and extensibility by Lisp code.

The example is an axiom written in KIF as follows:

```
(<=> (Type ?fp FunctionalProperty)
     (and (Type ?fp Property)
        (=> (and (PropertyValue ?fp ?s ?v1)
                 (PropertyValue ?fp ?s ?v2))
            (= ?v1 ?v2))))
```

All of the variables in this axiom are implicitly universally quantified. This axiom was, at one time, one of the axioms in the RDF ontology language. The axiom attempts to formalize the concept of a functional property, i.e., a property that takes exactly one value on every element of its domain, much as a mathematical function (or more precisely a partial function) does. It should exclude properties that are sometimes multi-valued.

The SNARK theorem prover very quickly found an inconsistency due to this axiom. The problem is that the axiom, as originally formulated, allows one to deduce that *every* property is functional. As a result, if a property is multi-valued, then one can deduce that some of its values must be equivalent. For example, the `rdf:type` property is heavily used by RDF and is highly multi-valued. For example, `rdf:Bag` is both of type `rdfs:Class` and `rdf:Resource` which implies that `rdfs:Class` is the same as `rdf:Resource`. Here is the output produced by SNARK at the conclusion of its refutation of consistency:

```
(Row 604 (= Class Resource) (resolve 407 406))
(Row 637 false (rewrite (paramodulate 70 604) 394))
```

The problem with the axiom above is that the variables `?s`, `?v1` and `?v2` should not be quantified using a global universal quantification. Their quantification must be scoped as follows:

```
(<=> (Type ?fp FunctionalProperty)
     (and (Type ?fp Property)
          (forall (?s ?v1 ?v2)
             (=> (and (PropertyValue ?fp ?s ?v1)
                      (PropertyValue ?fp ?s ?v2))
                 (= ?v1 ?v2)))))
```

SNARK also uncovered an inconsistency in the axioms for KIF, which served as a basis for the DAML axioms and which had been published since 1998 [5]. But probably the most far-reaching discovery SNARK made was that the axioms for the DAML cardinality restrictions were too weak to imply their intended consequences.

For example, in the Structured Walk-Through [15], a cardinality restriction was used to define a class (say `OneFather`) of elements with precisely one father. While it was possible to prove that an element of `OneFather` had at least one father, SNARK was unable to prove, because of the error in the axioms, that it could not have two fathers. As a result of SNARK's critique, the error was corrected in the revised axioms, which do imply the intended consequences.

## 5 Conclusions and Future Work

We have introduced the ConsVISor consistency checking tool that can be used to verify consistency of formal ontologies. A demonstration version is now available [8]. We are in the process of introducing new features to ConsVISor. One area of special concern is the problem of building more complex ontologies by merging

smaller ontologies. The smaller ontologies are distinct but overlap one another. When two ontologies are combined, it is necessary to specify how to mediate between concepts that are similar, but not entirely the same, in the ontologies. Inconsistencies are a common occurrence due to the differing assumptions and commitments made in the two ontologies, and ConsVISor can help to identify such problems and to correct them.

## References


1. G. Booch, J. Rumbaugh, and I. Jacobsen. *UML Notation Guide, Version 1.1*, September 1997.
2. Chimæra. Web site. `www.ksl.Stanford.edu/software/chimaera`.
3. DAML. DARPA Agent Markup Language Web Site, 2001. `www.daml.org`.
4. G. Frank. Hybrid reasoning architecture general purpose first-order logic theorem prover suite of special-purpose reasoners. `www.ksl.stanford.edu/software/JTP`.
5. M. Genesereth. Knowledge Interchange Format draft proposed American National Standard (dpANS) NCITS.T2/98-004, 1998. Available at `logic.stanford.edu/-kif/dpans.html`.
6. I. Horrocks. FaCT: Fast Classification of Terminologies Web Site. `www.cs.man.ac.uk/~horrocks/FaCT`.
7. Jess. Java expert system shell. `herzberg.ca.sandia.gov/jess`.
8. M. Kokar, J. Letkowski, K. Baclawski, and J. Smith. The ConsVISor consistency checking tool, March 2001. Available at `vis.home.mindspring.com/-consvisor.html`.
9. D. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, Breckenridge, Colorado, USA, April 12–15 2000.
10. OilEd. Ontology editor for DAML+OIL. `oiled.man.ac.uk`.
11. RDF. Resource description framework (RDF) model and syntax specification, Feburary 1999. `www.w3.org/TR/REC-rdf-syntax`.
12. SNARK. SRI's new automated reasoning kit. `www.ai.sri.com/~stickel/-snark.html`.
13. M. E. Stickel, R. J. Waldinger, and V. K. Chaudhri. A Guide to SNARK. `www.ai.sri.com/snark/tutorial/tutorial.html`.
14. UML Based Ontology Toolset. Web Site, 2001. `ubot.lockheedmartin.com`.
15. F. Harmelen van, P. Patel-Schneider, I. Horrocks, D. Connolly, L. Stein, and D. McGuinness, editors. *Annotated DAML+OIL Ontology Markup*. DARPA Agent Markup Language, March 2001. `www.daml.org/2001/03/daml+oil-walkthru.html`.