Declarative Evaluation of Ontologies with Rules

Dietmar Seipel, Joachim Baumeister, and Klaus Prätor

University of Würzburg, Institute of Computer Science, Germany
{seipel,baumeister}@informatik.uni-wuerzburg.de
praetor@mac.com

Abstract. Currently, the extension of ontologies by a rule representation is a very popular research issue. A rule language increases the expressiveness of the underlying knowledge in many ways. Likewise, the integration creates new challenges for the design process of such ontologies, but also existing evaluation methodologies have to cope with the extension of ontologies by rules. In this work, we introduce supplements to existing verification techniques to support the design of ontologies with rule enhancements, and we focus on the detection of anomalies that can especially occur due to the combined use of rules and ontological definitions.

Keywords. evaluation, anomalies, OWL, SWRL, RULEML, PROLOG, DATALOG

1 Introduction

The use of ontologies has shown its benefits in many applications of intelligent systems in the last years. It is not only a fantasy of computer scientists, but it correesponds to real needs. E.g., in *scholarly editions*, the lack of semantic search is very obvious. The works of the poet Stifter, e.g., are full of geological metaphors but the word *geological* is never mentioned. The philosopher Wittgenstein is dealing with philosophical problems, but does not use traditional philosophical terminology. So the editors are considering an *ontology* for the work of Wittgenstein. In the context of [PZW13], Pichler and Zöllner–Weber were also exploring the potential of PROLOG ontologies and logic reasoning as tools in the Humanities [Zoe09].

Whereas, the implementation of lower parts of the semantic web stack has successfully led to standardizations, the upper parts, especially rules and the logic framework, are still heavily discussed in the research community, e.g., see Horrocks et al. [3]. This insight has led to many proposals for rule languages compatible with the semantic web stack, e.g., the definition of SWRL (semantic web rule language) originating from RULEML and similar approaches [4]. It is well agreed that the combination of ontologies with rule–based knowledge is essential for many interesting semantic web tasks, e.g., the realization of semantic web agents and services. SWRL allows for the combination of a high–level abstract syntax for Horn–like rules with OWL, and a model theoretic semantics is given for the combination of OWL with SWRL rules. An XML syntax derived from RULEML allows for a syntactical compatibility with OWL. However, with the increased expressiveness of such ontologies, new demands for the development

and for maintenance guidelines arise. Thus, conventional approaches for evaluating and maintaining ontologies need to be extended and revised in the light of rules, and new measures need to be defined to cover the implied aspects of rules and their combination with conceptual knowledge in the ontology.

Concerning the expressiveness of the ontology language we focus on the basic subset of OWL DL (that should make the work transferable to ontology languages other than OWL) and we mostly describe syntactic methods for the analysis of the considered ontology. We also focus on the basic features of SWRL: we consider Horn clauses with class or property descriptions as literals, and we omit a discussion of SWRL built–ins. Due to the use of rules with OWL DL the detection of all anomalies is an undecidable task, cf. [4].

Here, the term verification denotes the syntactic analysis of ontologies for detecting anomalies. On one hand, the discussed issues of the presented work originate from the evaluation of taxonomic structures in ontologies introduced by Gómez–Pérez [5]. On the other hand, in the context of rule ontologies classical work on the verification of rule–based knowledge has to be reconsidered as done, e.g., by Preece and Shinghal [6, 7]. In their works, the verification of ontologies (mostly taxonomies) and rules (based on predicate logic), respectively, has been investigated separately. However, the combination of taxonomic and other ontological knowledge with a rule extension leads to new evaluation metrics that can cause redundant or even inconsistent behavior. The main contribution of our work is the extension of these measures by novel anomalies that are emerging from the combination of rule–based and ontological knowledge. Here, the concept of dependency graphs from deductive databases can be used [8]. Of course, the collection of possible anomalies may always be incomplete, since additional elements of the ontology language may also introduce new possibilities of occurring anomalies.

In detail, we investigate the implications and problems that can be drawn from rule definitions in combination with some of the following ontological descriptions: 1. class relations like *subclass of*, *complement of*, *disjointness* 2. basic property characteristics like *transitivity*, *ranges and domains*, and *cardinality restrictions*. We distinguish the following classes of anomalies:

- *Circularity* in taxonomies and rule definitions.
- *Redundancy* due to duplicate or subsuming knowledge.
- *Inconsistency* because of contradicting definitions.
- *Deficiency* as a category comprising subtle issues describing questionable design in an ontology.

The presented work is different from the evaluation of an ontology with respect to the intended semantic meaning: the OntoClean methodology [9] is an example for semantic checks of taxonomic decisions made in an ontology. We also do not consider common errors that can be implemented due to the incorrect understanding of logical implications of OWL descriptions as described by Rector et al. [12].

This paper is organized as follows: The next section gives basic definitions and describes the expressiveness of the underlying knowledge representation; in the context of this work a subset of OWL DL is used. Then, the four main classes of anomalies are discussed. In Section 3, we present a case study with anomalies in OWL ontologies. The paper is concluded with a discussion.

2 Expressiveness and Basic Notions

For the analysis of ontologies with rules, we restrict the range of the considered constructs to a subset of OWL DL: we investigate the implications of rules that are mixed with subclass relations and/or the property characteristics transitivity, cardinality restrictions, complement, and disjointness.

Given a class C and a property P. When used in rules, we call $C(x)$ a class atom and $P(x, y)$ a property atom. For the following it will be useful to extend the relations on classes and properties to relations on class and property atoms. Given two atoms A, A', we write $\odot(A, A')$, if both atoms have the same argument tuple, and their predicate symbols are related by $\odot$, i.e., if A and A' both are

- class atoms, such that $A = C(x)$, $A' = C'(x)$, and $\odot(C, C')$, or
- property atoms, such that $A = P(x, y)$, $A' = P'(x, y)$, and $\odot(P, P')$.

E.g., the relation $\odot$ can be `sub_class`, `isa`, `disjoint`, `complement`, etc. From a relationship $\odot(A, A')$ it follows that A and A' are of the same type.

2.1 Implementation in DATALOG*

The detection of anomalies has been done using a PROLOG meta–interpreter DATALOG*, which we have implemented in SWI PROLOG [13]. Due to their compactness and conciseness, we give the corresponding formal definitions for the anomalies, which are evaluated using a mixed bottom–up/top–down approach based on DATALOG and PROLOG concepts, respectively.

Variables such A, B, C, ..., A', or B_i can denote both class atoms and property atoms, whereas `As`, `Bs`, ..., denote sets of class atoms and property atoms. We denote a relationship A is-a A' by `isa(A, A')`. SWRL rules $B_1 \wedge \cdots \wedge B_n \Rightarrow A$ are represented as non–ground DATALOG* facts `rule(A-Bs)` (with variable symbols), where `Bs` $= [B_1, \ldots, B_n]$ is the list of body atoms and `A` is the head atom. Since SWRL rules with conjunctive rule heads can be split into several rules, we can – without loss of generality – assume rule heads are atomic. In DATALOG* and PROLOG rules, conjunction (and) is denoted by "`,`", disjunction (or) is denoted by "`;`", and negation by "`\+`".

Incompatible Classes: Complements and Disjointness. For classes, there exists the construct *complementOf* to point to instances that do not belong to a specified class. In DATALOG*, the complement relation between two classes `C1` and `C2` is denoted by `complement(C1,C2)`. In OWL, the disjointness between two classes is defined by the *disjointWith* constructor; with `disjoint(C1,C2)` we denote the disjointness between two classes `C1` and `C2`. We call two classes `C1` and `C2` *incompatible*, if there exists a disjoint or a complement relation between them. This is detected by the following PROLOG predicate:

```
incompatible(C1, C2) :-
    ( complement(C1, C2)
    ; disjoint(C1, C2) ).
```

Taxonomic Relationships and Rules An obvious equivalence exists between the relationships B is-a A – where A and B are both class atoms or both property atoms with the same arguments – and rules of the form $B \Rightarrow A$ with a single atom B in the body having the same argument as A. Thus, we combine them into the single formalism `derives` in DATALOG*:

```
derives(C1, C2) :-
   ( isa(C1, C2)
   ; rule(A-[B]) B =.. [C1, X1], A =.. [C2, X2],
     var(X1), X1 == X2 ).

isa(C1, C2) :-
   sub_class(C1, C2).
isa(C1, C3) :-
   isa(C1, C2), sub_class(C2, C3).
```

Observe, that the call `var(X1), X1 == X2` tests if `X1` and `X2` are bound to the same variable.

With the existence of equivalence definitions $E_1 \equiv E_2$ in an ontology language, e.g., the OWL definitions `equivalent_class` and `equivalent_property`, we can further extend the definition of `derives`: an element `E1` is derived by an element `E2`, if the elements are equivalent classes or properties. Since such an equivalence is symmetrical, the predicate `derives/2` always creates cyclic dervations of equivalent elements with length 1.

```
derives(E1, E2) :-
   ( equivalent_class(E1, E2)
   ; equivalent_property(E1, E2) ).
```

We compute the transitive closure `tc_derives` of `derives` using the following simple, standard DATALOG* scheme:

```
tc_derives(E1, E2) :-
   derives(E1, E2).
tc_derives(E1, E3) :-
   derives(E1, E2), tc_derives(E2, E3).
```

Subsequently, the reflexive transitive closure `tcr_derives` of `derives` is computed using the following PROLOG predicate:

```
tcr_derives(E1, E2) :-
   ( E1 = E2
   ; tc_derives(E1, E2) ).
```

Remark on Examples. In the following we give examples for most of the described anomalies. For this task, we use a printer domain, because to its popularity and intuitive understanding.

2.2 Mixing DATALOG and PROLOG: Forward and Backward Chaining

The detection of anomalies in SWRL ontologies could not be formulated using PROLOG backward chaining or DATALOG forward chaining alone, since we may need recursion on cyclic data, function symbols (mainly for representing lists), non–ground facts, negation and disjunction in rule bodies, aggregation, and stratification.

Thus we have developed a new approach that extends the DATALOG paradigm to DATALOG* and mixes in with PROLOG. However, an intuitive understanding of the presented, mixed rule sets is possible without understanding the new inference method. The interested reader can run the analysis using our DisLog system [2].

DATALOG*. We distinguish between DATALOG* rules and PROLOG rules. DATALOG* rules are forward chaining rules (not necessarily range–restricted) that may contain function symbols (in rule heads and bodies) as well as negation, disjunction, and PROLOG predicates in rule bodies. DATALOG* rules are evaluated bottom–up, and all possible conclusions are derived.

The supporting PROLOG rules are evaluated top–down, and for efficiency reasons only on demand, and they can in turn refer to DATALOG* facts. The PROLOG rules are also necessary for expressivity reasons: the are used for some computations on complex terms, and more importantly for computing very general aggregations of DATALOG* facts.

Ontology Evaluation in DATALOG*. For ontology evaluation, we have implemented two layers $\mathcal{D}_1$ and $\mathcal{D}_2$ of DATALOG* rules:

- The upper layer $\mathcal{D}_2$ consists of the rules for the predicate `anomaly/2` and some DATALOG* rules that are stated together with them.
- The lower layer $\mathcal{D}_1$ consists of all other DATALOG* rules. E.g., the rules for predicates `derives` and `tc_derives` are in $\mathcal{D}_1$.

$\mathcal{D}_1$ is applied to the DATALOG* facts for the following basic predicates, which have to be derived from the underlying SWRL document:

```
rule, class, sub_class, complement, incompatible,
equivalent_class, equivalent_property,
transitive_property, symmetric_property,
property_restriction, min_cardinality_restriction,
max_cardinality_restriction, class_has_property.
```

The resulting DATALOG* facts are the input for $\mathcal{D}_2$. The *stratification* into two layers is necessary, because $\mathcal{D}_2$ refers to $\mathcal{D}_1$ through *negation* and *aggregation*. Most PROLOG predicates in this paper support the layer $\mathcal{D}_2$.

E.g., the following predicates with calls to DATALOG* facts generalize `tc_derives` and `incompatible` to atoms:

```
tc_derives_atom(A1, A2) :-
   tc_derives(P1, P2), A1 =.. [P1|Xs], A2 =.. [P2|Xs].

incompatible_atoms(A1, A2) :-
   incompatible(P1, P2), A1 =.. [P1|Xs], A2 =.. [P2|Xs].
```

We cannot evaluate these rules using forward chaining, since `Xs` is a unknown list.

The head and body predicates of a rule can be determined using the following pure PROLOG predicates:

```
head_predicate(A-_, P) :-
   functor(A, P, _).

body_predicate(_-Bs, P) :-
   member(B, Bs), functor(B, P, _).
```

The following PROLOG rules define siblings and *aggregate* the siblings `Z` of a class `X` to a list `Xs` using the well–known meta–predicate `findall`, respectively:

```
sibling(X, Y) :-
   sub_class(X, Z), sub_class(Y, Z), X \= Y.

siblings(Xs) :-
   sibling(X, _),
   findall( Z,
      sibling(X, Z),
      Xs ).
```

These rules could also be evaluated in DATALOG* using forward chaining. But, since we need `siblings` only for certain lists `Xs`, this would be far to inefficient.

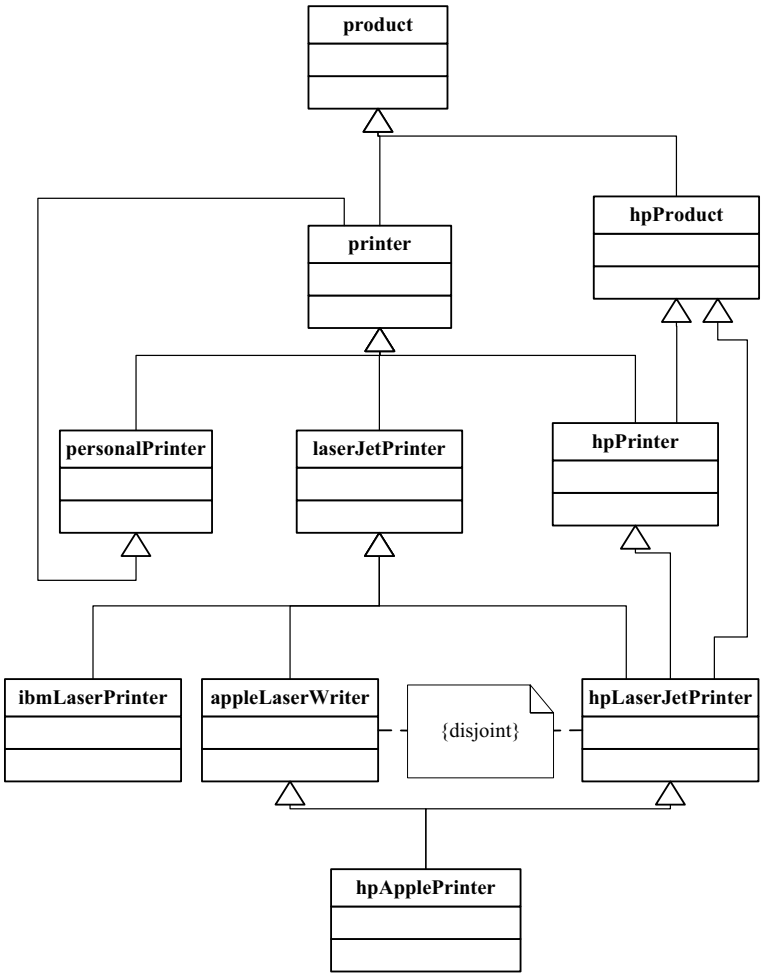
Evaluation of DATALOG*. DATALOG* rules cannot be evaluated in PROLOG or DATALOG alone for the following reasons: Current DATALOG engines cannot handle function symbols and non–ground facts, and they do not allow for the embedded computations, which we need. Standard PROLOG systems cannot easily handle recursion with cycles, because of non–termination, and are inefficient, because of subqueries that are posed and answered multiply. Thus, they have to be extended by some DATALOG* facilities (our approach) or memoing/tabling facilities (the approach of the PROLOG extension XSB). Since we wanted to use SWI PROLOG – because of its publicly available graphical API – we have implemented a new inference machine that can handle mixed, stratified DATALOG*/PROLOG rule systems.

3 Case Study

Knowledge representation in the Semantic Web is based on ontologies and logic. The reasoning tasks require *search* (query answering) and *knowledge engineering / modeling* (analysis of the structure of the ontologies for anomalies). Knowledge engineering and reasoning in the Semantic Web is based on ontology editors and specialized databases. It can further be supported by deductive databases and logic programming techniques.

In the Semantic Web, it is possible to reason about the ontology / taxonomy (i.e., the schema) and the instances. This is called terminological or assertional (T–Box or A–Box) reasoning, respectively. This makes *search* in the Semantic Web more effective.

– In the following printer ontology, we could search for a printer from HP, and the result could be a laser–jet printer from HP, since the system knows that `hpLaser-JetPrinter` is a sub–class of `hpPrinter`.
– It can also be derived, that all laser–jet printers from HP are no laser writers from Apple; in this case, this is very easy, since it is explicitely stored in the ontology.



Moreover, we will show in the following how to support *knowledge engineering* by detecting *anomalies* in OWL ontologies. In the *Web Ontology Language (*OWL*)*, we can mix concepts from `rdf` (Resource Description Framework) for defining instances and

rdfs (rdf Schema) for defining the schema of an application. Moreover, tags with the namespace owl are allowed. The Semantic Web Rule Language (SWRL) incorporates logic programming rules into OWL ontologies. There exist well–known, powerful tools for asking *queries* on and for *reasoning* with OWL ontologies.

3.1 The Printer Ontology in OWL

The following examples are given in Turtle syntax [15] using the namespace p for resources of the printer ontology. First of all, every laserJetPrinter is a printer, and every hpPrinter is an hpProduct:

```
p:printer         rdf:type owl:Class .
p:hpProduct       rdf:type owl:Class .
p:laserJetPrinter rdfs:subClassOf p:printer .
hpPrinter         rdfs:subClassOf p:hpProduct .
```

The following owl:Class element defines the class appleLaserWriter:

```
p:appleLaserWriter rdf:type owl:Class ;
    rdfs:comment "Apple laser writers are laser jet printers" ;
    rdfs:subClassOf p:laserJetPrinter ;
    owl:disjointWith p:hpLaserJetPrinter .
```

The rdfs:subClassOf sub–element states that appleLaserWriter is a sub–class of laserJetPrinter. The owl:disjointWith sub–element states that appleLaserWriter is disjoint from hpLaserJetPrinter.

The following owl:Class element defines a class of printers from a joint venture of HP and Apple:

```
p:hpApplePrinter
    rdfs:comment "Printers from a joint venture of HP and Apple" ;
    rdfs:subClassOf p:hpLaserJetPrinter, p:appleLaserWriter .
```

The existence of such printers would contradict the disjointWith restriction between the classes hpLaserJetPrinter and apperLaserWriter. The emptiness of the class hpApplePrinter can be detected by reasoners used, for instance, by ontology editors like Protégé.

Redundant subClassOf Relation. Since hpLaserJetPrinter is a sub–class of the class hpPrinter, and hpPrinter is a sub–class of hpProduct, it is redundant to explicitly state that hpLaserJetPrinter is a sub–class of hpProduct.

```
p:hpLaserJetPrinter
    rdfs:subClassOf p:laserJetPrinter, p:hpPrinter, p:hpProduct ;
    owl:disjointWith p:appleLaserWriter .
```

This redundancy is not an error. We could simply consider it as an anomaly, that should be reported to the knowledge engineer. This anomaly is usually not reported by reasoners in standard ontology editors.

Instances. Finally, we have some instances of the defined classes:

```
p:1001 rdf:type p:appleLaserWriter .
p:1002 rdf:type p:appleLaserWriter .
p:1003 rdf:type p:hpLaserJetPrinter .
p:1004 rdf:type p:hpLaserJetPrinter .
```

As mentioned before, there cannot exist instances of the class `hpApplePrinter`.

The ontology editor Protégé offers plugged–in reasoners, such as FaCT++, HermiT, and Racer. The ontology reasoner FaCT++ can inferr that the class `hpApplePrinter` is `EquivalentTo` the empty class `Nothing`. By clicking the question mark, an explanation can be shown. There are also databases for handling `rdf` data, so called triple stores, such as Sesame or Jena. They use extensions of SQL– most notably SPARQL – as a query language.

Please note, that for the presented Turtle syntax the corresponding XML syntax can be generated. For instance, the definition of the joint HP and Apple printer would read as follows:

```
<rdf:Description rdf:about="printer#hpLaserJetPrinter">
  <rdfs:subClassOf rdf:resource="printer#hpPrinter"/>
  <rdfs:subClassOf rdf:resource="printer#laserJetPrinter"/>
</rdf:Description>
```

Protégé. Figure 1 shows the printer ontology in the standard ontolgy editor Protégé.

3.2 Declarative Queries in FNQuery

In PROLOG, an XML element can be represented as a term structure `T:As:C`, called FN–triple. `T` is the tag of the element, `As` is the list of the attribute/value pairs `A:V` of the element, and `C` is a list of FN–triples for the sub–elements.

```
'owl:Class':['rdf:ID':'appleLaserWriter']:[
   'rdfs:comment':['Apple laser ...'],
   'rdfs:subClassOf':[
      'rdf:resource':'#laserJetPrinter']:[],
   'owl:disjointWith':[
      'rdf:resource':'#hpLaserJetPrinter']:[] ]
```

In an OWL knowledge base `Owl`, there exists an `isa` relation between two classes `C1` and `C2`, if a `subclassOf` relation is stated explicitely, or if `C1` was defined as the interesection of `C2` and some other classes:

```
% isa(+Owl, ?C1, ?C2) <-

isa(Owl, C1, C2) :-
   C := Owl/'owl:Class'::[@'rdf:ID'=C1],
   ( R2 := C/'rdfs:subClassOf'@'rdf:resource'
   ; R2 := C/'owl:intersectionOf'/'owl:Class'@'rdf:about' ),
   owl_reference_to_id(R2, C2).
```

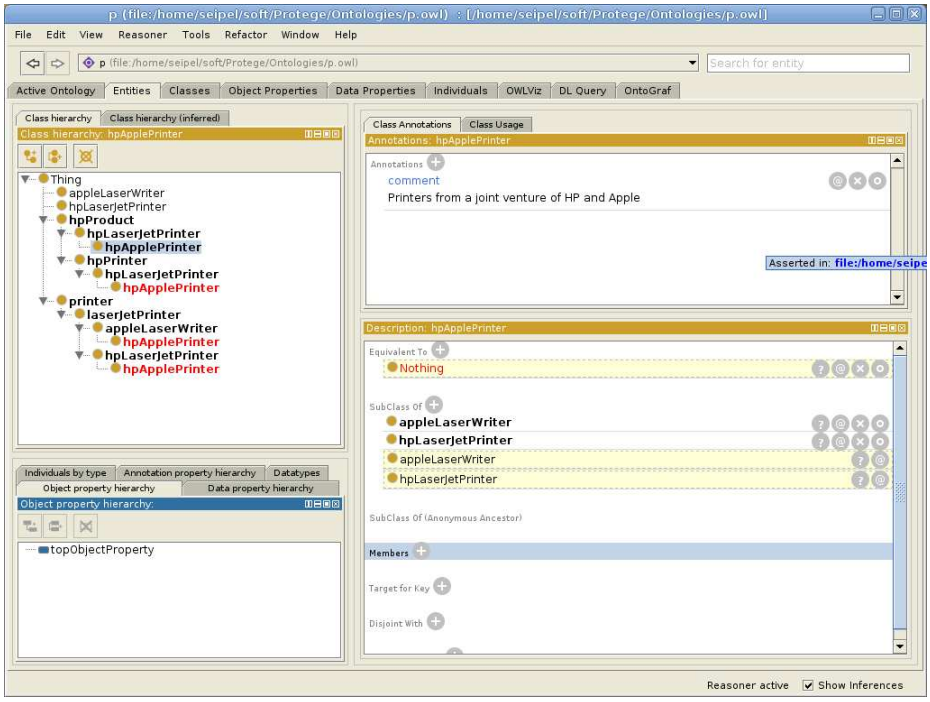


Fig. 1. The Printer Ontology in Protégé

```
% owl_reference_to_id(+Reference, ?Id) <-

owl_reference_to_id(Reference, Id) :-
   ( concat('#', Id, Reference)
   ; Id = Reference ).
```

Disjointness of Classes.

```
% disjointWith(+Owl, ?C1, ?C2) <-

disjointWith(Owl, C1, C2) :-
   R2 := Owl/'owl:Class'::[@'rdf:about'=R1]
      /'owl:disjointWith'@'rdf:resource',
   owl_reference_to_id(R1, C1),
   owl_reference_to_id(R2, C2).
```

In the following, we often suppress the ontology argument `Owl`.

Transitive Closure of `isa`.

```
% subClassOf(?C1, ?C2) <-

subClassOf(C1, C2) :-
   isa(C1, C2).
subClassOf(C1, C2) :-
   isa(C1, C), subClassOf(C, C2).
```

3.3 Anomalies in Ontologies

Cycle.

```
?- isa(C1, C2), subClassOf(C2, C1).

C1 = personalPrinter,
C2 = printer
```

Partition Error. The class `C` is a sub–class of two disjoint classes `C1` and `C2`.

```
?- disjointWith(C1, C2),
   subClassOf(C, C1), subClassOf(C, C2).

C  = hpApplePrinter,
C1 = hpLaserJetPrinter,
C2 = appleLaserWriter
```

Incompleteness. The class `C` has three sub–classes `C1`, `C2` and `C3`, from which only the two sub–classes `C1` and `C2` are declared as disjoint in the knowledge base.

```
?- isa(C1, C), isa(C2, C), isa(C3, C),
   disjointWith(C1, C2), not(disjointWith(C2, C3)).

C  = laserJetPrinter,
C1 = hpLaserJetPrinter,
C2 = appleLaserWriter,
C3 = ibmLaserPrinter
```

The fact that `C2` and `C3` are disjoint and that `C1` and `C3` are disjoint as well, possibly was forgotten by the knowledge engineer during the creation of the knowledge base.

Redundant subClassOf/instanceOf Relations. The sub–class relation between `C1` and `C3` can be derived by transitivity over the class `C2`.

```
% redundant_isa(?Chain) <-

redundant_isa(C1->C2->C3) :-
   isa(C1, C2), subClassOf(C2, C3),
   isa(C1, C3).

?- redundant_isa(Chain).

Chain = hpLaserJetPrinter -> hpPrinter -> hpProduct
```

Here, `isa(C1, C2), subClassOf(C2, C3),` requires that this deduction is done over at least two levels.

Undefined Reference. During the development of an ontology in OWL, it is possible that we reference a class that we have not yet defined.

```
% undefined_reference(+Owl, ?Ref) <-

undefined_reference(Owl, Ref) :-
   rdf_reference(Owl, Ref),
   not(owl_class(Owl, Ref)).

rdf_reference(Owl, Ref) :-
   ( R := Owl/descendant_or_self::'*'@'rdf:resource'
   ; R := Owl/descendant_or_self::'*'@'rdf:about' ),
   owl_reference_to_id(R, Ref).

owl_class(Owl, Ref) :-
   Ref := Owl/'owl:Class'@'rdf:ID'.
```

If we load such an ontology into Protégé, then the ontology reasoners may produce wrong results, even for unrelated parts of the ontology.

4 Discussion

In the last years ontologies have played a major role for building intelligent systems. Currently, standard ontology languages like OWL are extended by rule–based elements, e.g., RULEML and the semantic web rule language SWRL.

We have shown that with the increased expressiveness of ontologies – now also including rules – a number of new evaluation issues have to be considered. In this paper, we have presented a framework for verifying ontologies with rules comprising a collection of anomalies, that verify the represented knowledge in a combined methodology. For all anomalies, we have described a DATALOG* implementation which is used in a prototype for ontology verification. Due to its declarative nature, new methods for anomaly detection can be easily added to the existing work. From our point of view, the declarative approach is crucial because of the incompleteness of the presented anomalies: in principle, an entire overview of possible anomalies is not possible, since the number of anomalies depends on the used expressiveness of the ontology and the rule representation, respectively.

The actual frequency of the introduced anomalies is an interesting issue. However, only a small number of ontologies (mostly toy examples) is available that actually use a rule extension. A sound review of anomaly occurrences would require a reasonable number of ontologies having a significant size.

For many real–world applications, we expect a more expressive rule language to be used than SWRL. With SWRL FOL, an extension of SWRL to first–order logic is currently discussed as a proposal. Furthermore, larger systems may also include parts of a non–monotonic rule base. Here, some work has been done on the verification of non–monotonic rule bases [16], that has to be re–considered in the presence of an ontological layer.

References

1. J.Baumeister, D. Seipel: Anomalies in Ontologies with Rules.. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 8 (2010), No. 1, pp. 55–68.
2. D. Seipel, DisLog – A System for Reasoning in Disjunctive Deductive Databases, http://www1.pub.informatik.uni-wuerzburg.de/databases/DisLog/dislog_nmr.html.
3. I. Horrocks, B. Parsia, P. Patel-Schneider, J. Hendler, Semantic Web Architecture: Stack or Two Towers?, in: F. Fages, S. Soliman (Eds.), Principles and Practice of Semantic Web Reasoning (PPSWR), No. 3703 in LNCS, Springer, 2005, pp. 37–41.
4. I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, D. Tsarkov, OWL Rules: A Proposal and Prototype Implementation, Journal of Web Semantics 3 (1) (2005) pp. 23–40.
5. A. Gómez–Pérez, Evaluation of Ontologies, International Journal of Intelligent Systems 16 (3) (2001), pp. 391–409.
6. A. Preece, R. Shinghal, Foundation and Application of Knowledge Base Verification, International Journal of Intelligent Systems 9 (1994), pp. 683–702.
7. A. Preece, R. Shinghal, A. Batarekh, Principles and Practice in Verifying Rule-Based Systems, The Knowledge Engineering Review 7 (2) (1992), pp. 115–141.
8. S. Ceri, G. Gottlob, L. Tanca, Logic Programming and Databases, Springer, Berlin, 1990.
9. N. Guarino, C. Welty, Evaluating Ontological Decisions with OntoClean, Communications of the ACM 45 (2).
10. OWL 2: Web Ontology Language, Document Overview (Second Edition), W3C Recommendation, http://www.w3.org/TR/owl2-overview/ (December 2012).
11. A. Pichler, A. Zöllner–Weber, Sharing and Debating Wittgenstein by Using an Ontology, Wittgenstein Archives at the University of Bergen, Norway, Literary and Linguistic Computing 2013.
12. A. L. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, C. Wroe, OWL Pizzas: Practical Experience of Teaching OWL–DL: Common Errors & Common Patterns, in: Engineering Knowledge in the Age of the Semantic Web: 14th International Conference (EKAW), LNAI 3257, Springer, 2004, pp. 157–171.
13. J. Wielemaker, An Overview of the SWI-Prolog Programming Environment, in: Proc. of the 13th International Workshop on Logic Programming Environments (WLPE), 2003, pp. 1–16.
14. Y. Guo, Z. Pan, J. Heflin, LUBM: A Benchmark for OWL Knowledge Base Systems, Journal of Web Semantics 3 (2) (2005), pp. 158–182.
15. W3C, RDF 1.1 Turtle – W3C Recommendation, http://www.w3.org/TR/turtle/ (February 2014).
16. N. Zlatareva, Testing the Integrity of Non-Monotonic Knowledge Bases Containing Semi-Normal Defaults, in: Proc. of the 17th International Florida Artificial Intelligence Research Society Conference (FLAIRS), AAAI Press, 2004, pp. 349–354.
17. A. Zöllner–Weber, Ontologies and Logic Reasoning as Tools in Humanities, in: DHQ: Digital Humanities Quarterly 2009, Vol. 3, Nr. 4.