Open access • Book Chapter • DOI:10.1007/11891451_11

Verification and refactoring of ontologies with rules — Source link

Joachim Baumeister, Dietmar Seipel

Institutions: University of Würzburg

Published on: 02 Oct 2006 - Knowledge Acquisition, Modeling and Management

Topics: Semantic Web Stack, Code refactoring, Semantic Web and Ontology (information science)

Verification and Refactoring of Ontologies With Rules

Joachim Baumeister and Dietmar Seipel

Institute for Computer Science
University of Würzburg, Germany
email: {baumeister,seipel}@informatik.uni-wuerzburg.de

Abstract. Currently, the introduction of an appropriate rule representation layer for the semantic web stack is discussed. However, with the inclusion of rule-based knowledge new verification issues for rule-augmented ontologies arise.
In this paper we investigate the detection of anomalies as an important subtask of verification. We extend and revise existing approaches for the syntactic verification of ontologies with respect to the existence of rules, and we introduce new anomalies considering the understandability and maintainability of such ontologies.

1 Introduction

The use of ontologies has shown its benefits in many applications of intelligent systems in the last years. 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. [1].

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. 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 [2]. [1] SWRL allows for the combination of a high-level abstract syntax for Horn-like rules with OWL [4], 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.

In this paper, we revisit known approaches for the syntactic verification of ontologies and extend existing definitions with respect to rules if needed. Furthermore, we define novel measures detecting parts of the ontology that may create problems for the maintainability of the overall ontology. Such knowledge fragments are usually not

[1] Currently, SWRL [3] has the status of a W3C member submission.

responsible for inconsistencies, but their elimination often can improve the understandability and compactness of the ontology.

We focus on the basic features of SWRL and OWL, e.g., 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. [2]. Here, we only consider a subset of OWL DL, i.e., the combined use of rules with subclass relations and some property characteristics like transitivity, complement, and disjointness. In addition, we do not consider the evaluation of an ontology with respect to the intended semantical meaning, which for example is implemented by the OntoClean methodology [5] for taxonomic decisions made in an ontology. We also do not consider common errors that can be implemented due to mistakes with the logical understanding of OWL descriptions, e.g., as described by Rector et al. [6].

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 [7]. 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 [8,9]. 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. Here the concept of dependency graphs/relations from deductive databases can be used [10]. For the sake of simplicity we will use the term *ontology* with the meaning of *ontology with rules* in this paper.

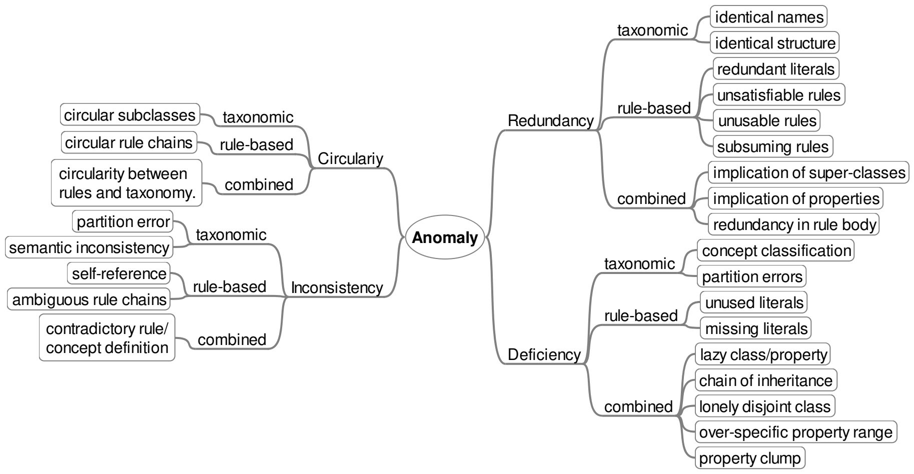

Fig. 1. A star of anomalies

We distinguish the following categories of anomalies: 1) *Redundancy* due to duplicate or subsuming knowledge in the ontology. 2) *Circularity* in taxonomies or rule definitions. 3) *Inconsistency* because of contradicting definitions. 4) *Deficiency* as a category comprising subtle issues affecting parts of an ontology with questionable design. Anomalies can occur for many reasons. For example, the integration of ontologies can yield redundant knowledge, and the manual development and evolution of a (large) ontology may introduce inconsistent definitions. Obviously, anomalies make the understandability, extensibility and evolution of ontologies more difficult. In Figure 1 the discussed anomalies are depicted as a star.

The elimination of anomalies is done by *refactoring*. This term originates from software engineering research [11,12], and it denotes the modification of source code without changing the external behavior of the program. The modification only focuses on the improvement of the code design rather than on its functionality. Analogously, the refactoring of ontologies should target the improvement of the design of the ontology, especially its understandability and maintainability.

The rest of the paper is organized as follows: Section 2 introduces the basic notions that are necessary for the analysis of ontologies with rules. In Section 3 we present measures to detect redundancy in ontologies, and in Section 4 variants of circularity are given. Section 5 describes the identification of syntactic inconsistency, and Section 6 introduces typical examples for deficient parts of the ontology and appropriate refactoring actions are sketched. Section 7 concludes the paper and gives directions for future work.

2 Basic Notions and Scope

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

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 $\circledast(A, A')$, if both atoms have the same argument tuple, and their predicate symbols are related by $\circledast$, i.e., if A and A' both are

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

For example, the relation $\circledast$ can be *is-a* , *disjoint*, *complementOf*, etc. Note, that from a relationship $\circledast(A, A')$ it follows that A and A' are of the same type.

The detection of anomalies has been implemented in SWI–PROLOG. Due to their compactness and formal manner we give the corresponding PROLOG definitions for the discussed anomalies. Rules $\beta \Rightarrow A$ are represented as `A-Body`, where `Body` is the list of body atoms (representing the conjunction β) 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 to be atomic.

2.1 Classes and Properties

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. Variables such as X, X', or X_i can denote both classes and properties, and A, A', or A_i can denote both class atoms and property atoms.

```
element(A) :-
   ( class(A)
   ; property(A) ).
```

In PROLOG, disjunction (or) is denoted by ";". Classes and properties are taxonomically related by *is-a* relations. In OWL such *is-a* relations are defined by *subClassOf* constructs. We denote a relation A *is-a* A' by `isa(A, A')`, where A, A' are either classes or properties.

2.2 Complements and Disjointness of Classes

For classes there exists the construct *complementOf* to point to instances that do not belong to a specified class. The complement relation between a class $C1$ and a class $C2$ is denoted by `complementOf(C1,C2)` in PROLOG.

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`. A set $\mathcal{C} = \{$`C1`$, \ldots,$`Cn`$\}$ of mutually disjoint classes defines a disjoint partition; in PROLOG we denote this by `disjointP([C1,...,Cn])`.

We call two classes $C1$ and $C2$ *incompatible*, if there exists a disjoint or (even) a complement relation between them.

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

2.3 Taxonomic Relations and Rules

Obviously, relations B *is-a* A – where A and B are both class atoms or both property atoms with the same arguments – are equivalent to rules of the form $B \Rightarrow A$ with a single atom B in the body, and we can combine the two into a single formalism $B \rightarrow A$. We denote the transitive closure of $\rightarrow$ by $\rightarrow^*$. In PROLOG, $B \rightarrow A$ can be described as follows:

```
derives(B, A) :-
   ( isa(B, A)
   ; rule(A-[B]) ).
```

In the following we will need implementations of the transitive closure of various predicates `<P>`, which all look like follows:

```
tc_<P>(A, C) :-
    ( <P>(A, C)
    ; <P>(A, B), tc_<P>(B, C) ).
```

I.e., for every predicate `<P>` for which we need the transitive closure we have a rule of the form above.[2] We will use the generic transitive closure for the predicate `isa`, where `tc_isa(A, A)` expresses that A is envolved in a cycle of the taxonomy (the *is-a* relation), and for the predicate `derives`.

3 Redundancy

Parts of the ontology are redundant due to duplicate definitions or subsuming definitions. Moreover, there could be redundant atoms in rule bodies, and the consequent of a rule could be unsatisfiable.

3.1 Identity

We call identical formal definitions of classes, properties or rules, that can be only discriminated by their different names, *identity errors*. They can occur if some implied knowledge is not explicitly stated in the ontology, thus uncovering an incompleteness error. For example, identically defined classes may be distinguished by the developer by the introduction of an additional property for one of the identical classes. Also identity of classes or rules can be created by the integration of overlapping ontologies that share (partially) identical concepts.

3.2 Redundancy by Subsumption between Rules

The redundant definition of taxonomic knowledge of classes and properties was already described by Gómez-Pérez [7]. Let X, Y be either two classes or two properties, such that X *is-a* Y is stated in the taxonomy. Then we distinguish *direct repetition*, where X *is-a* Y is stated more than once, and *indirect repetition*, where X *is-a* Y is stated and can at the same time also be derived by a chain X *is-a* X_1 *is-a* $\ldots$ *is-a* X_n *is-a* Y with $n \geq 1$. Direct and indirect repetition corresponding to the instantiation of classes and properties can be also defined on *instance-of* instead of *is-a* .

The redundancy of rule-based knowledge (in extended Horn clause representation) was considered for example by Preece and Shinghal [8]. A rule r is redundant with respect to the rule base, if for every environment (set of base facts) the exclusion of r would derive the same conclusions. In the following we define rule subsumption in general as well as two typical special cases.

[2] Note that a generic implementation of the transitive closure as a predicate `tc(<P>, A, C)` would of course be possible, but it would be less efficient, since the atoms `<P>(A, C)` and `<P>(A, B)` would have to be built repeatedly at run time.

Rule Subsumption. A rule $r = \beta \Rightarrow A$ subsumes another rule $r' = \beta' \Rightarrow$ A', if β subsumes β' and A subsumes A'. Then r fires more often than r' and derives more general consequences. This happens, e.g., if $A = A'$ and β' is a specialization of β.

```
anomaly(rule_subsumption, A1-Body1, A2-Body2) :-
   subsumes(Body1, Body2),
   subsumes([A1], [A2]).
```

This rule tries to instantiate A1-Body1 and A2-Body2 to A1'-Body1' and A2'-Body2', respectively, such that A1' subsumes A2' and Body1' subsumes Body2'. The instantiations generated in the call `subsumes(Body1, Body2)` are used in the subsequent call `subsumes([A1], [A2])`. There are two alternative implementations for the predicate `subsumes/2` depending on whether the first rule subsumes an instance of the second rule (partial subsumption), or it totally subsumes the second rule.

– The call `subset_non_ground(As1, As2)` in the first variant tries to instantiate As1 and As2 to As1' and As2', respectively, such that As1' is a subset of As2'. In that case As1 partially subsumes As2.

```
subsumes(As1, As2) :-
   subset_non_ground(As1, As2).
```

– In the second variant, before the call `subset_non_ground(As1, As2)` a copy As of As2 is made, which is afterwards compared to the new value of As2. If both are variants of each other, then As1 totally subsumes As2.

```
subsumes(As1, As2) :-
   copy_term(As2, As),
   subset_non_ground(As1, As2),
   variant(As2, As).
```

Implication of Superclasses. If A, A_i are either class or property atoms, then a rule $A_1 \wedge \cdots \wedge A_n \Rightarrow A$, such that $A_i \rightarrow^* A$ for some A_i, is redundant.

```
anomaly(implication_of_superclasses, A-Body) :-
   member(Ai, Body), tc_derives(Ai, A).
```

Here classes are only subsuming under certain conditions that are given in the rule condition, i.e., an incorrect assignment of the subclass relation may exist.

If $A_i \equiv A$, then the equivalence may be incorrectly assigned, since the rule condition denotes a restriction on the implication.

This can be seen as a special case of rule subsumption, since the fact $A_i \rightarrow^* A$ can be seen as a rule $A_i \Rightarrow A$, which subsumes the first rule given above.

Redundant Implication of Transitivity. If P is a transitive property, then a rule $P(x,y) \wedge P(y,z) \wedge \beta \Rightarrow P(x,z)$ embodies a redundant definition of P, which can be already derived by the OWL reasoner from the fact that P is transitive. Often such a redundancy can be explained by an erroneous assumption of the transitivity during an ontology integration process, since the rule defines a more restrictive condition of transitivity, if the conjunction β is non-empty.

```
anomaly(redundant_transitivity, P_xz-Body) :-
   P_xz =.. [P, X, Z],
   P_xy =.. [P, X, Y], P_yz =.. [P, Y, Z],
   subset_non_ground([P_xy, P_yz], Body).
```

This is a also special case of rule subsumption, since the transitivity of a property P can be expressed as a rule $P(x,y) \wedge P(y,z) \Rightarrow P(x,z)$, which subsumes the first rule given above.

3.3 Redundancy in the Antecedent of a Rule

For a rule $A_1 \wedge \cdots \wedge A_n \Rightarrow A$ we have $A_i \rightarrow^* A_j$ for two atoms in its antecedent. In this case the atom A_j is redundant and can be removed from the rule antecedent.

```
anomaly(redundancy_in_antecedent, A-Body) :-
   tc_derives(Ai, Aj),
   member(Ai, Body), member(Aj, Body).
```

As a special case, this form of redundancy can occur if $A_i \equiv A_j$ in the ontology. This anomaly may alternatively point to an incorrect mapping between the elements A_i and A_j.

3.4 Unsatisfiable Rule Condition

A rule has an unsatisfiable condition, if at least one literal neither unifies with an input literal (e.g., a given instantiation of the ontological concepts) nor with the consequent of another rule.

```
anomaly(unsatisfiable_condition, _-Body) :-
   member(A, Body),
   \+ fact(A),
   \+ rule(A-_).
```

With the rich semantics of OWL an unsatisfiable condition can also occur due to the contradictory use of *complementOf* or *disjointWith* descriptions.

```
anomaly(unsatisfiable_condition, _-Body) :-
   member(A, Body), member(B, Body),
   incompatible(A, B).
```

4 Circularity

Circular definitions in the ontology have a severe impact on the reasoning capabilities of the underlying knowledge. Here we distinguish circular definitions in the taxonomic structure of the ontology as described by [7], circular dependencies in the rule base as considered, e.g., by [8], but also circular dependencies that can occur due to the intermixture between taxonomic and rule-based knowledge.

Circularity in Taxonomy. There is a cyclic chain X_1 *is-a* X_2 *is-a* $\ldots$ *is-a* X_n, such that $X_1 = X_n$, where all X_i are classes or all X_i are properties.

```
anomaly(circularity_in_taxonomy, A) :-
   tc_isa(A, A).
```

Circularity between Rules and Taxonomy. There exists a rule $A_1 \wedge \cdots \wedge A_n \Rightarrow A$, such that for some atom A_i from the antecedent it holds $A \rightarrow^* A_i$.

```
anomaly(circularity_in_rules_and_taxonomy, A-Body) :-
   member(Ai, Body), tc_derives(A, Ai).
```

The specified rule should be considered as a restricted *is-a* relation between A and A_i, which may result in the detection of a misapplied taxonomic definition between the two concepts. This error is similar to *implication of subclasses*, but with an inverse *is-a* relation.

5 Inconsistency

Ambivalent definitions of ontological knowledge often cause unintended reasoning behavior. Besides partition errors concerning the taxonomic structure of the ontology, cf. [7], also ambivalent definitions within the rule base may occur, cf. [8]. However, due to the mixture of basic ontological knowledge and rules other ambivalence can be identified.

Partition Error in Taxonomy. Consider a disjoint partition of a class C into subclasses $C_1, \ldots, C_n$. On the class level, there is a partition error, if a class C' is a subclass of (at least) two disjoint subclasses C_i, C_j of C. On the instance level, a partition error, where some element e is an instance of (at least) two disjoint subclasses C_i, C_j of C, would lead to an inconsistency. The following rule defines a partition error on the class level:

```
anomaly(partition_error, A-[B, C]) :-
   disjoint(B, C),
   isa(A, B), isa(A, C).
```

Self–Contradicting Rule. For a rule $A_1 \wedge \cdots \wedge A_n \Rightarrow A$ there exists a *complementOf* or a *disjointWith* relationship between A and one of its body atoms A_i. Note that, according to our definitions in Section 2, this means that $A = C(x)$ and $A_i = C_i(x)$ are class atoms with the same argument x, and that C and C_i are disjoint or complements.

```
anomaly(contradicting_rule_consequent, A-Body) :-
   member(B, Body), incompatible(A, B).
```

If such a rule would fire, then the derived conclusion A of the rule would contradict the assumption A_i in its antecedent.

Contradicting Rules. We say that a rule $r = \beta \Rightarrow A$ contradicts another rule $r' = \beta' \Rightarrow A$', if β subsumes β', but A and A' are contradicting. If r' would fire, then also the stronger r would fire and the derived conclusions A' and A would be contradicting. The subsumption β subsumes β' can be defined by *equivalentClass/Property* relations as well as *is-a* relations. The consequents $A = C(x)$ and $A' = C'(x')$ are contradicting, if the corresponding classes C and C' are disjoint or complements.

```
anomaly(ambivalent_rule_pair, A1-Body1, A2-Body2) :-
   incompatible(A1, A2),
   subsumes(Body1, Body2).
```

An even more general form of the anomaly is given, if there are two sets of rules (not necessarily disjoint) that are deriving two semantically contradicting conclusions.

6 Deficiency

Deficiency is a subtle category comprising anomalies in an ontology that neither can be identified as redundant nor define inconsistent knowledge. Such anomalies can originate from the manual development of (large) ontologies, the evolution of ontologies, or as a side-effect of the integration of ontologies. Deficiency is usually not responsible for reasoning errors but affects the completeness, understandability or maintainability of the underlying knowledge.

Originally, such *design anomalies* had been identified and investigated for relational databases. In the last years, software engineering research has coined the term *bad smells* for parts of the source code that do not produce false behavior but are badly designed and should be improved for better maintainability, cf. [11]. Recently, a first step was taken to transfer this idea to the conceptual properties of rule-based knowledge [13] and OWL ontologies [14], respectively.

The identification of a bad smell is the starting point of a *refactoring*. Refactoring methods describe precise procedures to eliminate the corresponding smell without changing the meaning of the remaining knowledge. The following measures can be only seen as indicators for the occurrence of an anomaly. In any case the user has to decide whether and how to remove the possible anomaly. Then, refactoring methods provide constructive procedures that restructure the ontology and rule base by eliminating the anomaly.

In the following we present heuristics for the identification of some design anomalies, and we sketch the use of appropriate refactoring methods.

6.1 Lazy Class/Property

An element (class or a property) in the ontology that is actually never used in the real-world application is called *lazy*. The following facts indicate that an element could be lazy:

- the element represents a leaf in the hierarchy,
- no rules use this element,
- there exist no instances of the element.

Laziness can occur due to many reasons: The merge or the integration of two ontologies may include terms that are not useful or relevant in the actual domain. In addition, an element can evolve to be lazy if it was specialized or generalized to elements more appropriate to the application domain; consequently, the element was kept in the ontology although it is not used anymore. In PROLOG, a possibly lazy element A can be detected as follows:

```
anomaly(lazy_element, A) :-
   element(A),
   \+ isa(_, A),
   \+ in_rule(A),
   \+ instance(_, A).

in_rule(A) :-
   rule(H-Body),
   ( A = H
   ; member(A, Body) ).
```

The constraints stated above can be relaxed by tolerating very few rules with the considered object in their head or body. Then, these rules have to be inspected by the user and marked as not usable any more. Removing the unused element with the refactoring *delete element* should be considered with reasonable care:

1. The hierarchy has to be reconnected, i.e., every child of the term has to be linked as a child to every parent of the element.
2. The attributions of the term have to be reattached to its children, e.g., transitivity for a lazy property.
3. The corresponding rules have to be edited, i.e., every rule that contains the element either in its antecedent or in its consequent has to be reconsidered: rules with the element in their consequent should be removed from the ontology. Rules with the element in their antecedent are either removed (default for rules with the element as the only literal in the antecedent) or changed (remove literal with the element from the antecedent). For the latter we have to consider the creation of anomalies,

such as the creation of redundant or ambivalent rules. In any case, changed rules should be presented to the developer for a manual revision.

6.2 Chains of Inheritance

The backbone of an ontology is described by classes with corresponding taxonomic relations, i.e., classes are hierarchically connected by *is-a* relations. If ontologies are manually build in a distributed environment or are developed by the integration using parts of other ontologies, then the indented subclass structure can degenerate to *is-a* cascades in some areas of the taxonomy.

A taxonomic chain

$$C_1 \textit{ is-a } C_2 \textit{ is-a } \ldots \textit{ is-a } C_n \, ,$$

of classes C_i, such that all intermediate concepts $C_2, \ldots, C_{n-1}$ are contained in no other *is-a* relations except the ones in the chain is called a *chain of inheritance*. The following observations for these intermediate classes C_i can be used as a heuristic to strengthen the suspicion that a chain is anomalous:

– there exist no or very few instances for the C_i,
– the C_i are not extensively used in rules or other ontological definitions, e.g., property restrictions

In any case the user has to decide if the chain should be eliminated by the refactoring *collapse hierarchy*. Then, the chain is reduced to

$$C_1 \textit{ is-a } C_n \, ,$$

and the intermediate concepts C_i ($2 \leq i \leq n-1$) are subsequently removed from the ontology as follows:

1. All *properties* where C_i occurs as the domain, as the range, or in a restriction have to be modified. In many cases the occurrence of C_i can be changed to the upper class C_1. But in some cases these properties may appear to be redundant or useless, then the property should be considered to be removed as well.
2. All *rules* containing C_i have to be modified. If there exist many rules containing C_i in the antecedent or consequent, then the refactoring may not be practical. However, a reasonable heuristic may be to change the occurrences of C_i to C_1, if $2 \leq i \leq n/2$, or to C_n, if $n/2 < i \leq n-1$, i.e., to change the class to the nearest remaining neighbor.
3. For all *instances* of C_i – similar to the handling of rules – the user has to decide if the existing instances should be translated to instances of C_1 or C_n.

Finally, a new subclass relation C_1 *is-a* C_n, which replaces chain, is created, and the classes $C_2, \ldots, C_{n-1}$ are removed from the ontology.

6.3 Lonely Disjoint Class

The anomaly *lonely disjoint class* can occur as a result of an ontology integration task. A lonely disjoint class is a concept that is not disjoint with any of its siblings, but has disjoint relations to a collection of classes that are mutual siblings in another branch of the taxonomy.

```
anomaly(lonely_disjoint, C) :-
   siblings(Cs),
   disjointP([C|Cs]),
   \+ ( sibling(C, M), disjoint(C, M) ).
```

Besides an integration task such a lonely disjoint class can also occur due to the manual modification of the ontology, i.e., moving a class into another branch without the subsequent adaptation of the disjoint relations.

If the user has classified the disjoint relation as an actual error, then the elimination of this anomaly is quite simple: the disjoint property can be removed from the lonely disjoint class. However, its existence can cause unindented reasoning behavior.

6.4 Over-Specific Property Range

Developers tend to be very specific when manually defining value ranges for the particular properties. For example, the value range of a property *temperature* may be

$$R_{\textit{temperature}} = \{\ \textit{very high}, \textit{high}, \textit{normal}, \textit{low}, \textit{very low}\ \}.$$

During the practical use of the ontology it might turn out that the values are too specific and that the coarser value range $R'_{\textit{temperature}} = \{\ \textit{high}, \textit{normal}, \textit{low}\ \}$ would work much better. If rules are defined containing this property, then the anomaly can be identified by the existence of many analogous rules for the particular values. In our example, rules for the values *high* and *very high* could be present. In such cases, the refactoring *coarsen value range* forms groups of equivalent values, e.g., *high' =* { *high, very high* } and *low' =* { *low, very low* }.

The following rule determines pairs of rules having variants *has_value(P, Vi)*, i=1,2, of property values in their antecedent (after deleting these variant atoms their bodies are identical):

```
anomaly(over_specific,
      R1, R2, has_value(P, [V1, V2])) :-
   rule(R1), rule(R2),
   R1 = A1-Body1, R2 = A2-Body2, R1 \= R2,
   delete(has_value(P, V1), Body1, B1),
   delete(has_value(P, V2), Body2, B2),
   A1-B1 = A2-B2.
```

An analogous rule can be stated for rule consequents. The refactoring also replaces the original values with the aggregated ones in the corresponding rules, which is illustrated by the following example.

For the automatic *refactoring* of the corresponding rules the developer needs to define a mapping $M : R_{\textit{temperature}} \mapsto R'_{\textit{temperature}}$ from the original range to the coarsened range, e.g.:

v	very high	high	normal	low	very low
M(v)	high	high	normal	low	low

Every rule containing the property *temperature* is refactored by the application of the mapping function. Every atom in the head or body with *has_value(temperature, v)* is replaced by another atom *has_value(temperature, v')*, where *v' = M(v)*. Analogously, we have to replace all values in OWL constructs where values are explicitly used, e.g., in *hasValue* property restrictions.

With the application of the refactoring *coarsen range* redundant rules may be produced. In the case of a semantically inconsistent mapping function map even inconsistent rules can occur. In consequence, the existence of such anomalies has to be checked in a subsequent step.

6.5 Property Clump

The manual and distributed development of a larger ontology or the integration of existing ontologies can produce unintentionally repeated definitions in different classes of the ontology.

A *property clump* is a set $\mathcal{C}$ of classes having a relatively large set $\mathcal{P}$ of properties in common. These properties include the instantiation of DataType properties and Object properties.

For *refactoring*, the repeated use of the property clump $\mathcal{P}$ can be caught by a new class $C_{\mathcal{P}}$, which gets the properties in $\mathcal{P}$. The original classes $C \in \mathcal{C}$ are linked to $C_{\mathcal{P}}$ instead of linking them to the properties in $\mathcal{P}$. For ontologies with rules, we have to change all rules having property atoms $P(x, y)$ for $P \in \mathcal{P}$ in their antecedent or consequent.

The use of such an abstract property class $C_{\mathcal{P}}$ may increase the compactness and the maintainability (with respect to chances, extensions, fixes) of the ontology. A property clump in ontologies is comparable to the repeated use of code fragments in traditional software, so-called *clones*. The extraction of such repetitions into a single method or data structure is a common refactoring, which improves the compactness and maintainability of the code. The procedure of the corresponding refactoring *extract concept* is sketched for ontologies by the following example.

Example (Extract Concept for Property Clump)

The repeated definition of the *String* DataType properties

$$\mathcal{P} = \{\, \mathit{hasAddress}, \mathit{hasPhone}, \mathit{hasEmail} \,\}$$

having the classes $\mathcal{C} = \{\, \mathit{person}, \mathit{company} \,\}$ as domain can be aggregated to a new concept $C_{\mathcal{P}} = \mathit{addressInfo}$. If the user decides that the aggregation of these properties is a meaningful self-contained concept, then the refactoring *extract concept* can automatically perform the following steps:

1. Create a new class $C_{\mathcal{P}} = $ *addressInfo* and add the class *addressInfo* as a new possible domain for all identified properties in $\mathcal{P}$.
2. Create a new object property *hasAddressInfo* connecting the classes $C \in \mathcal{C}$ with the new class *addressInfo*, where $\mathit{range}(\mathit{hasAddressInfo}) = \{\mathit{addressInfo}\}$ and

   $$\mathit{domain}(\mathit{hasAddressInfo}) = \bigcup_{P \in \mathcal{P}} \mathit{domain}(P).$$

3. Create and redirect instances: For each instance of a class in $\mathcal{C}$, create an appropriate instances of class *addressInfo* and property *hasAddressInfo* and redirect the original properties in $\mathcal{P}$ with respect to the newly created property *hasAddressInfo* and class *addressInfo*.
4. Change rules having properties $P \in \mathcal{P}$ in their antecedent. E.g., for the property *hasAddress(X,Y)* a new rule is created

   $$\mathit{hasAddressInfo}(C, C') \wedge \mathit{hasAddress}(C', A) \Rightarrow \mathit{hasReAddress}(C, A) \qquad (1)$$

   and every original rule, e.g.,

   $$\mathit{hasAddress}(C, A) \wedge \mathit{hasLoc}(A, L) \Rightarrow \mathit{hasAdrLoc}(C, L) \qquad (2)$$

   is changed to

   $$\mathit{hasReAddress}(C, A) \wedge \mathit{hasLoc}(A, L) \Rightarrow \mathit{hasAdrLoc}(C, L) \qquad (3)$$

   We see that for each property $P \in \mathcal{P}$ a new rule (1) is created, and the property P of the original rule (2) is replaced by the newly created property which was defined in rule (1), see adapted rule (3). This is reasonable if the property is used in many rules and these rules should be kept as compact as possible. Otherwise, we would encode the redirection in the original rules, i.e., instead of rule (1) and rule (3) we simple would modify rule (2) by exchanging the atom $\mathit{hasAddress}(C, A)$ in the antecedent by the conjunction

   $$\mathit{hasAddressInfo}(C, C') \wedge \mathit{hasAddress}(C', A),$$

   which enlarges the antecedent by one literal.
5. Change rules having properties $P \in \mathcal{P}$ in the consequent. For example, the rule

   $$\mathit{hasLoc}(C, L) \wedge \mathit{hasAdrLoc}(L, A) \Rightarrow \mathit{hasAddress}(C, A) \qquad (4)$$

   with *hasAddress(C, A)* in the consequent is replaced by the rule

   $$\begin{aligned} \mathit{hasLoc}(C, L) \wedge \mathit{hasAdrLoc}(L, A) \wedge & \mathit{hasAddressInfo}(C, C') \\ & \Rightarrow \mathit{hasAddress}(C', A) \end{aligned} \qquad (5)$$

7 Conclusions and Future Work

The implementation of the semantic web stack currently focuses on the integration of rules into the web ontology language OWL. At the moment, SWRL, the semantic web rule language, is a proposal for such an integration, and it has the status of a W3C member submission. With the use of rule-based knowledge in combination with taxonomic definitions of the ontology, new evaluation questions arise. In consequence, evaluation measures have to be revisited and extended in order to include rules.

In this paper, we have presented a revised approach for the verification of rule augmented ontologies which also includes extended measures for the verification of ontologies with respect to more subtle anomalies concerning the understandability and maintainability. With the description of the anomalies we also sketched appropriate *refactoring methods* for eliminating the detected problems.

In general, the work is not limited to the expressiveness of the SWRL ontologies, but it can be also applied to similar rule extensions of ontologies. However, the presented approach is only a starting point for an extensive framework for the verification and refactoring of ontologies. Here, only parts of the expressiveness of OWL DL were considered; e.g., the implications of possibly existing property restrictions (universal and existential quantification, cardinalities) are not investigated in the presented work. Besides the consideration of the full expressiveness of OWL DL and of SWRL and its extensions, e.g., to first-order logic by SWRL FOL [15], we also need to consider the availability of *non-monotonicity*, which is expected to play an important role in real life ontologies and knowledge bases. Here, some work has been done on the verification of non-monotonic rule bases [16], that should be also integrated in a more elaborated framework.

References

1. Horrocks, I., Parsia, B., Patel-Schneider, P., Hendler, J.: Semantic Web Architecture: Stack or Two Towers? In Fages, F., Soliman, S., eds.: Principles and Practice of Semantic Web Reasoning (PPSWR). Number 3703 in LNCS, SV (2005) 37–41
2. Horrocks, I., Patel-Schneider, P.F., Bechhofer, S., Tsarkov, D.: OWL Rules: A Proposal and Prototype Implementation. Journal of Web Semantics **3**(1) (2005) 23–40
3. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language - Combining OWL and RuleML, W3C Member Submission . `http://www.w3.org/Submission/SWRL/` (May 2004)
4. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference – W3C Recommendation. `http://www.w3.org/TR/owl-ref/` (Feb. 2004)
5. Guarino, N., Welty, C.: Evaluating Ontological Decisions with OntoClean. Communications of the ACM **45**(2) (2002)
6. Rector, A.L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: 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) 157–171
7. Gómez-Pérez, A.: Evaluation of Ontologies. International Journal of Intelligent Systems **16**(3) (2001) 391–409
8. Preece, A., Shinghal, R.: Foundation and Application of Knowledge Base Verification. International Journal of Intelligent Systems **9** (1994) 683–702
9. Preece, A., Shinghal, R., Batarekh, A.: Verifying Expert Systems. A Logical Framework and a Practical Tool. Expert Systems with Applications **5(3/4)** (1992) 421–436

10. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer, Berlin (1990)
11. Fowler, M.: Refactoring. Improving the Design of Existing Code. Addison-Wesley (1999)
12. Opdyke, W.F.: Refactoring Object-Oriented Frameworks. PhD thesis, University of Illinois, Urbana-Champaign, IL, USA (1992)
13. Baumeister, J., Seipel, D., Puppe, F.: Refactoring Methods for Knowledge Bases. In: Engineering Knowledge in the Age of the Semantic Web: 14th International Conference, EKAW, LNAI 3257, Springer (2004) 157–171
14. Baumeister, J., Seipel, D.: Smelly Owls – Design Anomalies in Ontologies. In: Proc. of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS), AAAI Press (2005) 215–220
15. Patel-Schneider, P.F.: A Proposal for a SWRL Extension to First-Order Logic. `http://www.daml.org/2004/11/fol/proposal` (Nov. 2004)
16. Zlatareva, N.: 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) 349–354