ONTORULE

Ontologies meet Business Rules

D2.3 Consistency Maintenance. Intermediate Report

Jörg Pührer (TUWIEN)

with contributions from:

Adil El Ghali (IBM), Amina Chniti (IBM), Roman Korf (ontoprise GmbH), Antonia Schwichtenberg (ontoprise GmbH), François Lévy (Paris13), Stijn Heymans (TUWIEN), Guohui Xiao (TUWIEN), Thomas Eiter (TUWIEN)

Abstract.
One goal in WP2 is to investigate and develop techniques for detecting when the creation or modification of rules and/or the ontology affects the overall consistency. Moreover, techniques to help the business user or the rule/ontology manager repair detected inconsistencies, including new capabilities such as testing and debugging across the rule/ontology boundary should be investigated and developed. This is an intermediate report on consistency maintenance in which we setup a general framework of consistency maintenance tasks, including definitions, detection, and handling of consistency related problems. Methods for natural language and individual rule, ontology, and combined formalisms are presented in terms of the framework.

Keyword list: consistency maintenance, consistency in texts, inconsistency, anomaly, Object-Logic, Description Logics, DL-programs, diagnosis, debugging

Work package number and name	WP2 Business Rules and Ontologies Ownership and Management
Deliverable nature	Report
Dissemination level	Public/PU
Contractual date of delivery	M24
Actual date of delivery	January 21, 2011

ONTORULE Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number 2009-231875 .

ILOG/IBM
9, rue de Verdun
94253 Gentilly Cedex
France
Tel: +33 1 49 08 29 81
Fax: +33 1 49 08 35 10
Contact person: Mr. Christian de Sainte Marie
E-mail: csma@fr.ibm.com

Ontoprise GmbH
An der RaumFabrik 29
76227 Karlsruhe
Germany
Tel: +49 721 50980910
Fax: +49 721 50980911
Contact person: Mr. Jürgen Angele
E-mail: angele@ontoprise.de

Free University of Bozen-Bolzano
Faculty of Computer Science
Piazza Domenicani 3
I-39100 Bozen-Bolzano BZ, Italy
Italy
Tel: +39 0471 016 120
Fax: +39 0471 016 009
Contact person: Mr. Enrico Franconi
E-mail: franconi@inf.unibz.it

Technische Universität Wien
Institut für Informationssysteme
AB Wissensbasierte Systeme (184/3)
Favoritenstrasse 9-11
A-1040 Vienna
Austria
Tel: +43 1 58801 18460
Fax: +43 1 58801 18493
Contact person: Prof. Thomas Eiter
E-mail: eiter@kr.tuwien.ac.at

PNA Training B.V.
Geerstraat 105
6411 NP Heerlen
The Netherlands
Tel: +31 455600 222
Fax: +31 455600 062
Contact person: Prof. Sjir Nijssen
E-mail: sjir.nijssen@pna-training.nl

Université de Paris 13
LIPN
99, avenue J.B. Clément
F-93430 Villetaneuse
France
Tel: +33 1 4940 4089
Fax: +33 1 4826 0712
Contact person: Prof. Adeline Narazenko
E-mail: adeline.narazenko@lipn.univ-paris13.fr

Fundación CTIC
Edificio Centros tecnológicos
33203 cabueñes - Gijón
Asturias
Spain
Tel: +34 984 29 12 12
Fax: +34 984 39 06 12
Contact person: Mr. Antonio Campos
E-mail: antonio.campos@fundacionctic.org

Audi
Auto Union Strasse
D-85045 Ingolstadt
Germany
Tel: +49 841 89 39765
Contact person: Mr. Thomas Syldatke
E-mail: thomas.syldatke@audi.de

ArcelorMittal
Marques de Suances
33400 - Avilés
Spain
Tel: +34 98 5126 404
Fax: +34 98 5126 375
Contact person: Mr. Nicolas de Abajo
E-mail: nicolas.abajo@arcelormittal.com

Executive Summary

This document presents state-of-the-art and intermediate results on consistency maintenance in the ONTORULE project. A general consistency framework is provided as a basis for communicating consistency related requirements and methods for the individual formalisms considered within the project. The framework defines common terminology, classifies consistency related problems, and identifies main tasks for consistency maintenance. On the problem side, four abstract problem types for classification of consistency problems are presented. Regarding tasks for consistency maintenance, we distinguish methods for diagnosing consistency problems and actions to overcome them.

We give an overview of the state of the art for consistency maintenance in texts, logical rules, production rules, and Description Logics, placing its different components in the general framework as appropriate.

As regards inconsistency arising from texts in natural language, we focus on the acquisition setting in ONTORULE, i.e., the scenario where rules are extracted from plain text. Two main issues are addressed. On the one hand the case that the output of vocabulary and rule extraction is inconsistent because of erroneous interpretation of the text and on the other hand the case that the formulation of statements in the text itself potentially cause inconsistency. A main objective is prevention of inconsistencies during acquisition.

We describe a declarative approach towards detecting inconsistencies and anomalies in ObjectLogic language which originates from the F-Logic rule language. The method identifies typical cases of inconsistency using ObjectLogic itself. Moreover, methods for handling the identified problems are discussed.

For consistency maintenance in production rules, we adapt an approach using design patterns for ontology management. We introduce Change Management Patterns (CMP) that classify different types of changes, inconsistencies, and possible solutions. The main idea is that when a rule set is changed, the corresponding patterns determine which types of inconsistencies need to be checked, and which solution alternatives are available.

We give a state-of-the-art summary on research into inconsistency in Description Logics. Here, the focus is on problems related to logical contradictions, i.e., inconsistency in the sense of logical unsatisfiability. We classify consistency problems in Description Logics and deal with techniques for recognizing inconsistencies. Here, we discuss approaches that aim at explaining inconsistencies to a knowledge engineer, as well as approaches

that aim at pinpointing parts of the knowledge base that causes an inconsistency. In the discussion of actions for handling inconsistencies, we deal with consistency restoring methods on the one hand, and inconsistency tolerant approaches on the other hand.

Finally, initial results on consistency maintenance in combinations of ontologies and rules are reported. Here, the focus is on loosely coupling of Description Logics and logical rules realized by DL-programs. Results carry over to the related language of F-Logic# which was developed within WP3. We address two consistency related phenomena that may emerge from the features of the coupling mechanism. We provide definitions and complexity analysis for diagnosing minimal sets of calls to the Description Logic knowledge base that can restore consistency of inconsistent DL-programs. Moreover, we present a novel semantics for DL-programs that tolerates inconsistencies caused by the ontology update-feature of the loose-coupling mechanism.

Table of Contents

Chapter 1

Introduction

In this deliverable we present intermediate results on consistency maintenance in the ONTORULE project. We provide a general consistency framework in order to establish comparability between consistency related requirements and methods for the individual formalisms considered within the project. The framework, which is introduced in Chapter 2, defines common terminology, classifies consistency related problems, and identifies main tasks for consistency maintenance. In the subsequent chapters we discuss consistency maintenance for texts, logical rules, production rules, and Description Logics, respectively. We give an overview of the state of the art placing its different components in the general framework. Moreover, we provide new methods for consistency maintenance for loosely-coupled combinations of Description Logics and logical rules.

The deliverable is organized as follows. In Section 2.1 we lay down a unified terminology to speak about knowledge bases in rule, ontology, and combined languages. Besides consistency within the deliverable, the motivation for that is that a common terminology can reveal similarities in consistency problems and solutions between the different formalisms. In Section 2.2 four abstract problem types for classification of consistency problems are presented.

Chapter 3 deals with inconsistency arising from texts in natural language. The basic scenario is that rules are extracted from plain text which is a central setting in ONTORULE WP1 “Business policy acquisition and modelling”. Two main issues are addressed, on the one hand the case that the output of vocabulary and rule extraction is inconsistent because of erroneous interpretation of the text and on the other hand the case that the formulation of statements in the text itself potentially cause inconsistency. A main objective is avoiding such problems already in the acquisition phase. Hence, before problem analysis and consequent actions in the production phase are discussed in Section 3.2, we discuss prevention of inconsistencies during acquisition in Section 3.1.

In Chapter 4 we consider consistency maintenance in logical rules. In particular, we deal with the ObjectLogic language which originates from F-Logic. We study the formalism along the lines of our framework. In Section 4.1 we present typical cases of inconsistency. Moreover, in Sections 4.2 and 4.3 methods for problem identification, respectively problem handling are presented that extend work for DATALOG.

Analogously, Chapter 5 deals with production rules, discussing common problems in Section 5.1, problem diagnosis in Section 5.2, and actions towards problem solving in Section 5.3. Here, we introduce Change Management Patterns as approach for consistency maintenance.

Chapter 6 gives a state-of-the-art summary on research into inconsistency in Description Logics. Here, the focus is on problems related to logical contradictions, i.e., inconsistency in the sense of logical unsatisfiability. After giving preliminaries on Description Logics in Section 6.1, following the scheme of the consistency framework, we classify consistency problems in Section 6.2. Section 6.3 deals with techniques for recognizing inconsistencies. Here, we discuss approaches that aim at explaining inconsistencies to a knowledge engineer, as well as approaches that aim at pinpointing parts of the knowledge base that causes an inconsistency. In the discussion of actions for handling inconsistencies in Section 6.4, we deal with consistency restoring methods on the one hand, and inconsistency tolerant approaches on the other hand.

In Chapter 7 we report initial results on consistency maintenance in combinations of ontologies and rules. Here the focus is on loosely coupling of Description Logics and logical rules realized by DL-programs. Results carry over to the related language of F-Logic# which was developed within WP3 “Execution and inference”. Section 7.1 provides basic notions for DL-programs. We address two consistency related phenomena that may emerge from the features of the coupling mechanism as discussed in Section 7.2.2. We provide definitions and complexity analysis for diagnosis of inconsistent DL-programs in Section 7.3. Moreover, in Section 7.4 we present a novel semantics for DL-programs that tolerates inconsistencies caused by the ontology update-feature of the loose-coupling mechanism. Chapter 8 concludes the report.

Chapter 2

General Consistency Framework

In this chapter we define a general framework for consistency maintenance in different formalisms important in the ONTORULE project such as logical rules, production rules, and Description Logics. It should serve as a basis for communicating work on consistency related issues within ONTORULE Task 2.4. We identify three areas in the process of consistency maintenance and state their basic questions:

1. **Problems.** What are the consistency problems that a production or logical rule set, a DL axiom set, a combination of those,[1] can have? E.g., a consistency problem could be that a rule is not applicable (the body can never be true or will never trigger).
2. **Diagnosis.** Why is a certain consistency problem applicable to our knowledge base? Two questions arise here:
   (a) **Checking.** How to check the problem against the knowledge base? This might happen in a *statical way*, i.e., by analyzing the knowledge base syntactically (e.g., no running of its native reasoner) or in a *dynamic way*, i.e., by specifically running the native reasoner against the, possibly modified, knowledge base.
   (b) **Explanation.** After checking the problem, what constitutes an explanation? An explanation might have different *granularities* and might need to satisfy different *conditions*, e.g., minimality of an explanation (if a set of atoms A is explaining a problem then a set of atoms $B \supset A$ is not a minimal explanation).
3. **Actions towards Problem Solving.** Using the diagnosis can the problem be fixed? How to fix it?

[1]When talking about any of these possibilities, we will refer to them as the knowledge base.

2.1 Basic Definitions

In order to indicate the different problems that are relevant for consistency maintenance, we introduce some general terminology. For a particular knowledge base KB, we distinguish between its *assertional knowledge*, denoted by $\mathcal{A}(KB)$, and its *terminological knowledge*, denoted by $\mathcal{T}(KB)$. For example, in case KB represents a logical rule knowledge base, $\mathcal{A}(KB)$ would consist of facts (rules with an empty body, i.e., an empty `IF` condition) and its terminological knowledge would consist of rules with non-empty body. For Production rules the distinction is similar, where the assertional knowledge is a particular scenario (the input space) and the terminological knowledge is formed by the rules (the mapping from the input space to the decision space). For Description Logics the terminological knowledge consists of the TBox and the assertional knowledge of an ABox. For combination approaches the distinction is similar. For example for a DL+PR combination, the assertional knowledge would consist of the ABox of the DL and the input space of the PR.

For the individual paradigms we will introduce appropriate notions of an *interpretation* for a knowledge base such that the semantics of the formalisms can be defined by the concept of a *model*. Moreover, for the different approaches we will use a *consequence operator* $\models$ such that $M \models KB$ holds whenever an interpretation M is a model of the knowledge base KB. The precise meanings of $\models$ and M will be specified later. However, for example, a model of an LP knowledge base could refer to an answer set in case of Answer-Set Programming or to a well-founded model in the case of F-Logic Programming. For Production rules this will be a pair of input space and decision space where the attributes have been modified by the rule actions triggered by the input space. In Description Logics, it will refer to its standard first-order model. In combinations, it will refer to whatever is defined as a model in that context.

For rule-based formalisms, we make use of the notion of *applicability* of a rule. Conditions will be given defining under which conditions a rule $r \in KB$ in a knowledge base KB is applicable w.r.t. an interpretation I of KB.

For example, a production rule is applicable if, given the input space of I, the rule's `IF` part matches. A logical rule is applicable if its `IF` part is true w.r.t. a given interpretation. A DL axiom is applicable if its left hand side is true in the given interpretation.

2.2 Abstract Definitions of Problems

Given the terminology introduced in the previous section, we now define abstract categories of consistency problems.

- **Contradiction.** We distinguish between two types to indicate when a knowledge base KB is *contradictory*:

  - Terminological contradiction: KB does not have a model. E.g., for a DL Concept A with non-empty domain, the unsatisfiable axiom $A = \neg A$ leads to a global contradiction.
  - Assertional contradiction: KB has a model, but there is assertional knowledge A' such that $\mathcal{T}(KB) \cup A'$ does not have a model. For example, a set of rules and a particular input space would lead to a model, but a different input space with the same set of rules would not. Or, a DL knowledge base with a TBox and ABox would have a model, but changing the ABox would lead to not having a model. For a concrete example in LP, consider the facts

    ```
    joan:Woman.        joan:Pope.
    ```

    as the assertional part and the constraint

    ```
    !- ?x:Woman AND ?x:Pope.
    ```

    as terminological part, expressing that women are currently not allowed to be pope. The overall KB would not have any model.

- **Completeness.** For completeness, we assume for a knowledge base KB, the existence of a set of pairs (T', α) where $(T' \subseteq \mathcal{T}(KB))$, i.e., a subset of the terminological knowledge of KB, and α is a set of interpretations for KB. We call this set of pairs the *completion set* of KB, where choosing an adequate completion set is subject to the formalism and the problem at hand. A knowledge base KB is then *complete* if for each of its completion pairs (T', α) and for each $I \in \alpha$ there is a rule in T' that is applicable by I. A small example is given as follows. Assume $\mathcal{T}(KB)$ consists of the rules

  $$\texttt{IF } \mathit{score} < 50 \texttt{ THEN } \mathit{category} = \mathit{Silver}$$

  and

  $$\texttt{IF } \mathit{score} >= 65 \texttt{ THEN } \mathit{category} = \mathit{Platinum}.$$

  Then, for a score between 50 and 65 category is undefined.

- **Relevance.** We distinguish between three problems and call a knowledge base KB *relevant* if:

  - for each $r \in \mathcal{T}(KB)$ one has some given interpretation I such that r is applicable w.r.t. I, e.g., the (production) rule

    $$\texttt{IF } \mathit{year} = 1848 \texttt{ THEN } \mathit{category} = \mathit{Platinum}$$

    is irrelevant if the domain of year are numbers greater than 1950.
  - all $r \in \mathcal{T}(KB)$ have *effect*, where the meaning of a rule having effect remains to be defined for individual approaches.

  - $\mathcal{T}(KB)$ contains no *redundancy*. Again, what is redundancy is up to the respective formalism in consideration. E.g., in LP, the second of the two rules

$$a \leftarrow b \qquad a \leftarrow b, c$$

    would contain redundancy as from b alone a would follow.
  - there is no other form of *language specific relevance*. For example, for loosely-coupled approaches, a query to the ontology might become irrelevant if it causes an inconsistency in that ontology (e.g., each entailment-based query is true if it makes the ontology inconsistent)

- **Language Conformant.** This collects sets of problems that are specific to the modeling language used, i.e., considering its methodology, computational properties, etc. For example, updating a knowledge base might lead to falling out of a desired fragment of the language, e.g., one might make rules unsafe or recursive (e.g., adding a rule `a :- b.` if there is already a rule `b :- a.`) without that the reasoner can handle this, or where it is undesired to have fragments that are more expressive for efficiency reasons (even though the reasoner would be able to handle them). Language conformance also holds *best practices* for a particular language, e.g., type correctness in F-Logic.

2.3 Diagnosis and Actions towards Problem Solving

Recognizing problems that might occur and methods for solving them are provided specifically for the individual approaches in the following chapters. Regarding combinations of formalisms our approach will be to reduce inconsistency handling as much as possible to the cases of the individual formalisms.

Chapter 3

Consistency and texts

The analysis of requirements in natural language has given rise to recent efforts ([1], [11], [15], [28], [29]). The objective of these papers is mainly to describe a translation process of a faithfull text into some logical form. Several papers suppose that the source is written in a controlled language. Only [23] considers the quality of the source, but it obeys this restriction. The authors of [55] aim at detecting defaults in specifications through logical inconsistencies in their formal model.

When vocabulary and rules are extracted from plain texts, the question of inconsistency extends beyond its logical part. Once a logical inconsistency is discovered, whatever its category, the question is, as described in chapter 2, to diagnose and repair the problem. Suppose some rule is considered as a possible culprit; then the cause can be searched at three levels:

- either the output of extraction has been erroneously translated during the authoring process,
- or this output is by itself incorrect, due to an erroneous interpretation of the text,
- or the authors of the text did not realize that some statements could yield inconsistencies.

The last two cases are in the scope of this chapter. According to the category of inconsistency, tools created for acquisition and related data structures are also used in order to help clarifying the cause. It must be noted that checking at this acquisition level if a specific interpretation is erroneous, is not a formal task and is not automatically feasible. The approach considered is to focus on relevant parts of text, so helping an expert to quickly and accurately discover where is the problem.

But authoring with an imperfect description of the rules, discovering an inconsistency and tracing back through the interpretation is a rather costly process. Another strategy is first considered, which aims at detecting the cause upstream, during the interpretation phase, preventing the incorrect output. The method is rather innovative: to the best of our knowledge, searching for problems at the source level without imposing severe controls on its vocabulary and syntax has not received attention for the moment. Here again, automatic discovery is not feasible, but focusing on adequately chosen subsets can help the acquisition expert to improve the quality of the output.

3.1 Prevention

The objective is to prevent inconsistencies to be introduced during the knowledge acquisition phase, or to detect them if they are already present in the source text. As the knowledge in question is in the process of being formalized, logical verifications are not available. The techniques proposed rely on the index structure: text has received annotations attaching ontological labels and rule qualifications to text fragments, and these fragments are linked to the ontology, the rule base and together, thus allowing to the links and select fragments according to various criterion. The base of inconsistency detection is a search engine performing this selection through a request which can mix literal elements and labels. The whole set of data structures and tools is described in [40].

We now turn to the different cases of inconsistency and requests which can bring them into light.

3.1.1 Filtering some cases of contradictions

Contradictions can arise due to different kinds of clumsy formulations, apart from the direct formulation of one fact and its negation.

Inconsistent statements of a property

It may rather easily happen that a property is phrased differently at two different places, and that basic world knowledge makes the literal translations of both descriptions inconsistent. The following example belongs to the terms and conditions of a loyalty program:

> *The membership year, which is the period in which your elite benefits are available, runs from March 1 through the last day of February of the following year.*
>
> *Since you begin receiving elite benefits immediately upon qualification, the beginning of your membership year is the day you qualify for your elite level.*

If the modeler makes the interpretation that a period runs from its beginning to its end, then qualifying another day as March 1 will yield a contradiction. This is in fact the correct interpretation in most cases, but it must be understood that this definition does not hold upon qualification, and the second one holds in its place.

Grouping sentences which involve the defined term can help solving the problem. In the particular case, the concept *membership year* occurs four times along 10 pages. The last one, after two more pages, is disambiguated:

> *Your membership year (March 1 or the date you qualify for elite status, through the last day of February).*

Rules are inconsistent in some particular cases

It may also happen that different restrictions exclude through their cumulative effect a case which was intended to be accepted. Here is an example from the 'speaker agreement' proposed by organizers of a congress to tutorial speakers:

> *Each tutorial speaker receives a grant of $500.00 to assist with his or her travel to the conference. A maximum of two speakers per tutorial (or $1000 per tutorial) will be awarded travel grants.*

Since ≪$1000 per tutorial≫ can be read as an emphasis on the *two speakers* limit, the text logically excludes more than two speakers, which was not intended by the organizers: it had been sent to the four speakers of an accepted tutorial. To solve the difficulty, it must be understood that the rule only applies to tutorials having at most two speakers, and that otherwise it is $1000 per tutorial - the grand differing then from $500.

While the agreement is by itself too short to valid experiments, it may be noticed that the two sentences of interest contain "speaker", "grant(s)" and "travel", and that, in a significant corpus, a query for sentences involving the three concepts should gather interesting fragments.

Exceptions are implicit

An important reason why contradictions easily arise in interpreting texts is the frequent use, in natural language, of unexplicited exceptions. The following fragments are extracted from the same text and are sparsed along ten pages, so their relations, which are nearly obvious when they are gathered, require a very carefull reading to be brought to light:

> *(1) On single-plane flights, you'll receive the nonstop origin-destination mileage.*
>
> *(2) If you are an Elite member, you will earn a minimum of 500 miles on applicable routes.*

> *(3) Certain airline tickets are not eligible for mileage credit.*
>
> *(4) You'll receive AAdvantage mileage credit only for the class of service on which your fare is based when you are ticketed.*
>
> *(5) As an elite-status member you earn an elite status mileage bonus on the base or guaranteed minimum miles for each eligible flight.*

Sentence (1) seems to define the benefit for any member on any flight. In (2), it is learned that the definition is different for a subcategory. In (3), that it is not universal with respect to flights. (4) adds the information that the benefit varies with the class of service, an apparent contradiction with (1). And (5) adds that it varies also with the status, through a *bonus* mechanism.

All these sentences contain a lexical variant of the concept *mileage benefit*, and a verbal form expressing either transfer ("earn", "receive"), or the right of transfer ("be eligible"). They are obtained through an and/or query on the index, the verbal forms being indexed as roles. Such a query is of course not guaranteed to give exactly the relevant set of sentences, but can provide a good upper approximation.

It must be noted that implicit exceptions are present even in legal texts. For instance, the United Nation regulation number 16 states:

> *The forward displacement of the manikin shall be between 80 and 200 mm at pelvic level in the case of lap belts. In the case of other types of belts, the forward displacement shall be between 80 and 200 mm at pelvic level and between 100 and 300 mm at chest level.*

but the next paragraph continues

> *In the case of a safety-belt intended to be used in an outboard front seating position protected by an airbag in front of it, the displacement of the chest reference point may exceed that specified in paragraph 6.4.1.3.2. above if its speed at this value does not exceed 24 km/h.*

and other exceptions are considered in the following.

3.1.2 Checking the completeness of cases

This kind of inconsistency arises when some case of an ordered list is not dealt with by the rules while all others are, so it seems that an information has been missed. The ordered list of cases reflects a domain which is described by this list, and which must be covered by rules. E.g., if fares depend on the age, each age must be caught by a rule computing its fare. Or if different test methods can be used according to the type of the belt assembly, each type must be caught by a rule providing the adequate method.

Based on domain

A first way to check for completeness at the textual acquisition level rely on the ability to mark domains and properties to be covered, for instance the age of the passenger, the type of belt assembly or its type of retractor.

The method makes the assumption that, in the representation of rules, a premise and a conclusion part have been recognized. Checking can be performed by gathering similar rules, in the sense that their premises are variants one of another, i.e., involve the same property with a different value in the domain, and are identical with respect to other elements. The identity constraint can be relaxed to a selected subset of the premise.

This method allows to check that, since there is a rule related to the extraction conditions of a strap equipped with a manually unlocking retractor, the same conditions are dealt with in similar form for other types of retractors, namely for non-locking retractors, for automatically locking retractors and for emergency locking retractors,

Based on conclusion

It may happen that rules related to different cases are not organized the same, so they are not textual variants. A second strategy relies on similarity of conclusions: for instance gathering the rules which conclude to a reduced fare, or in the safety belt testing example to a distance between locking positions.

This strategy may be too tolerant and produce so many rules that they need to be filtered again. For instance, there may be many other conditions justifying reduced fares, beside the age. On the other hand, the distance between locking positions only depends on the locking mechanism. And this strategy eases understanding that the rules are incomplete w.r.t. non-locking retractors because these don't need to test properties of their locking positions.

3.1.3 Filtering some redundant cases

A rule is redundant in a set of rules if it can be deleted without changing the conclusions. Redundancy cannot be recognized at the textual level with such a generality. We propose to restrain to simple one to one comparisons of rules which might be performed on the fly when a new rule is discovered.

One premise more precise than the other, same conclusion

A first form of redundancy is when the conclusions of two rules are identically indexed, but all the ontological elements in the premise of one (the less restrictive) also belong to the premise of the other. If indexing is correct, the second one is redundant. Note that this is clearly the case when ontological vocabulary and knowledge are stable. In the first stages of textual acquisition, it may be the case that some domain concept is lacking or incompletely indexed. In this case, the comparison may help to improve the ontology.

Same premise, conclusion more precise

The second form is symmetric, considering the possibility that different rules applicable to a given situation are stated at different places and may happen to be redundant. The same caveat applies w.r.t. the stability of ontological knowledge.

Redundancy and justifications

If redundancy at the execution level is preferably pruned for efficiency reasons, care must be taken of the justifications – i.e., the links from the text to the output passed to the authoring task. Regulations can change, and the masking rule may be abolished independently of the redundant one. For maintenance, generally, the justifications must be preserved.

3.2 Diagnosis and actions

Now we consider a different schema: the rules are in use and a problem is discovered at the application level. Then it is traced back to the implemented rules at its source, and then to textual rules involved, to help diagnose how the problem happened and decide how to repair.

To trace back, links between text fragments and the resulting rules have to be preserved. In the tradition of diagnosis theory, they are called justifications. At the difference of the previous section, ontological knowledge is stable and all the rules have been detected. The basic techniques remain analogous.

3.2.1 Conflict between executable rules

The executable layer provides a set of rules in conflict, and the justifications incriminate a set of text fragments to consider again. The cause of the problem may be either that the business rules are by themselves ill formed, or that their interpretation during the knowledge acquisition phase has been deficient.

Ill-interpreted text

The causes for an incorrect interpretation have been described in Section 3.1.1, e.g., the implicit exception problem (3.1.1), or an ambiguous phrasing. Once the problem identified, justifications allow a partial redo of the knowledge acquisition.

Uninterpretable rules

Another case is that the text can hardly be interpreted with certainty. [34] gives the following fragment of the Norwegian National Insurance Act. In the act, it is stated that residency in Norway gives a membership in the National Insurance. Residency is defined by the text:

> *Norwegian resident is defined as one who is staying in Norway, when the stay is intended to last or has surpassed 12 months. A person who moves to Norway is considered a resident from the date of arrival.*

Supposing that the second sentence applies only if the stay is intended to last 12 months - either it would be in conflict with he preceding one - and that *intended to last* has received a precise criterion, the question remains of medication refunding when the criterion was fulfilled at the date of arrival, but the stay was interrupted: according to the date when the decision is taken, opposite answers can obtain.

In such a situation, the solution is to turn back to the authors of the text or some person having authority to reformulate or clarify it.

3.2.2 Incompleteness

Suppose that, at execution time, a missing case is discovered, *i.e.*, a case was presented, for which no answer could be computed. The information is rather fuzzy, since any information in the input may be the cause of the problem. The situation is better if comparable cases are available - either archived or artificially generated ones. Cases which differ on a single data and get an answer are of particular interest: this data may be the bearer of incompleteness. Let is be called a *blocker* of the case. It is always possible to systematically test if a variation of one single data allows an answer, to obtain a list of blockers.. Analogous information might also be obtained from tracing at the execution level.

Note that justifications are not here of a great help: since no rule is applicable, none is directly identified as a culprit. The question is rather that some rule is lacking. On the other side, the blockers need not be unique. The text can be searched for fragments with all the ontological elements of the case- query yielding the least set of fragments - or for fragments with a blocker - the largest set of fragments - as for intermediate queries, looking if a fragment reveals an untranslated rule.

It is also interesting to consider what is obtained when the blocker is modified, because, by analogy, one of these information might be in the conclusion of the missing rule. The query may then be focused on fragments containing one of these elements.

Chapter 4

Consistency in Logical Rules

The aim of this chapter is to describe the general strategy of building a framework for detecting inconsistencies and other anomalies within ObjectLogic [46], [47] the successor of F-Logic [2], [12].

We use a declarative approach in order to verify ontologies combined with rules at the symbolic level. The theory of the here proposed approach is described mainly in [7]. Instead of using DATALOG [33] for describing axioms detecting the anomalies we use ObjectLogic.

The general idea is to write axioms that define the circumstances an inconsistency or some other anomaly might occur. In order to write these axioms we need information about the entities in the ObjectLogic modules. Modules here can be seen as the ObjectLogic counterpart of the term ontology in OWL. Since some of the information occurs in many axioms it will be implemented in separate rules to be reused. Some of the information as e.g. how ObjectLogic rules are assembled need to be described on a meta level to make assumptions on the ObjectLogic elements.

The first section introduces various possible anomalies arising within ObjectLogic knowledge bases. The second section will introduce various methods in order to identify such issues. The last section will then give hints on how to solve these issues.

The list of anomalies is not intended to be complete, but rather a first selection of anomalies that might be of relevance within the scope of the ONTORULE project. The main approach is adopted from [7], [6], [58] and [50].

4.1 Problems

In this section we introduce various possible issues arising within ObjectLogic knowledge bases. Not all of them are real issues but might hint on bad-practice or not intended modeling practice. We are reusing the structure of the general consistency framework introduced in chapter 2 in order to structure the different anomalies.

4.1.1 Contradiction

Ambivalent Rules

A rule base contains ambivalent rules if these rules infer facts that contradict some constraints. This anomaly has been described within [50]. Within ObjectLogic it is possible to define constraints on the knowledge base. Constraints within ObjectLogic define circumstances under which the intended knowledge base is violated. Constraints can be posed as queries. A constraint is violated if it yields results.

The example below contains a constraint stating that a *Student* cannot be *Teacher* at the same time. However the rule concludes this constraint violation.

```
Student::Person.
Teacher::Person.
Lecture[teacher*=>Person].
john:Student.
math:Lecture[teacher->john].

!- ?x:Student AND ?x:Teacher.

?x:Teacher :- ?x:Person AND ?l:Lecture AND
     ?l[teacher->?x].
```

Ambivalent Rule Pairs (Contradicting Rules)

A rule base contains ambivalent rule pairs R_1 and R_2 if the body B_2 of a rule R_2 subsumes the body B_1 of a rule R_1 and their consequents infer incompatible facts. The anomaly has been described in [50] and [7].

The example below contains the disjoint concepts *Student* and *Teacher*. The concepts are disjoint due to the constraint stating that a instance cannot be *Student* and *Teacher* at the same time. It also contains two rules while the first rule body subsumes the body of the second rule. However the conclusions of the rules are contradicting due to the violation of the constraint. Due to the rules all instances derived as *Teacher* would also be derived as *Student*.

```
Lecture[attends *=> Person].
Lecture[teacher *=> Person].

!- ?x:Student AND ?x:Teacher.
```

```
?x:Student :- ?x:Person AND ?l:Lecture[attends->?x].
?x:Teacher :- ?x:Person AND ?l:Lecture[attends->?x]
     AND ?l[teacher->?x].
```

Incompatible Rule Antecedent

This anomaly occurs if there are incompatible relationships between two body literals B_1 and B_2 of a rule R. This could be e.g. a disjoint or complement relationship. The anomaly has been described in [7].

The example below contains a rule that states that some instances belonging to the concepts *Teacher* and *Student* at the same time are instance of the concept *PhDStudent*. However, the constraint states that there cannot be a instance belonging to both concepts *Teacher* and *Student*.

```
?x:PhDStudent :- ?x:Student AND ?x:Teacher.

!- ?x:Student AND ?x:Teacher.
```

Self-Contradicting Rule

A rule R is self-contradicting if it contains a head literal H_i and body literal B_j that are incompatible. In this case we assume that the literals are concept assertions that are disjoint or complement. The anomaly has been described in [7].

The following example presents a self-contradicting rule. The concepts *Student* and *Teacher* are disjoint due to the constraint stating that an instance cannot be instance of *Student* and *Teacher* at the same time. However, the rules contains the head literal *?x:Teacher* and the body literal *?x:Student* which violates the constraint.

```
Lecture[teacher *=> Person].

?x:Teacher :- ?x:Student AND
     (EXIST ?l ?l:Lecture[teacher->?x]).

!- ?x:Student AND ?x:Teacher.
```

4.1.2 Completeness

Undefined Concept

According to [58] the habit of using undefined concepts within instantiations is not a good practice. A concept is undefined either if:

- It does not have sub-concepts or super-concepts
- The schema information does not define properties for the concept
- It is not declared as concept.

In the following example the instance *John* is instantiated from the concept *Student*. However the concept *Student* is not defined within the small knowledge base.

```
John:Student.
```

Concept Leaf

The anomaly occurs if a concept has neither instances, nor sub-concepts nor is it used in a rule literal. The anomaly has been defined by [58].

In the following very small example the concept *Student* in not further used within one of the relations described above.

```
Student::Person.
```

Rule Property Undefined

According to [58] the anomaly exists if a rule uses a property that is not defined in a signature atom.

In the following example the property *teaches* is used within the rule but it is not defined within the knowledge base.

```
Teacher[].
Student[].
Lecture[].

?x:Teacher :- ?x:Student AND
     (EXIST ?l ?l:Lecture[teacher->?x]).
```

4.1.3 Relevance

Redundancy by Repetitive Taxonomic Definition

In [7] the authors define two types of these issues. The first is the *direct repetition*. This redundancy occurs when the same axiom is stated more than once. A small example the sub-concept relation for *A* and *B* is stated more than once should demonstrate this kind of anomaly.

```
A::B.
a:A.
A::B.
```

This is usually not an issue within the ObjectLogic implementation of OntoBroker. OntoBroker will treat the redundant information as one. Therefore it cannot be detected during runtime within OntoBroker.

However the *indirect repetition* can exist due to chains of sub-concept relations.

```
A::B.
A::B1.
...
Bn-1::?Bn.
Bn::B.
```

The example below defines a direct sub-concept relation between *Student* and *Person* but also two indirect sub-concept relations between them via a chain: *Student* is sub-concept of *Man* and *Woman* and both are sub-concept of *Person*.

```
Person[].
Student::Person.
Student::Man.
Student::Woman.
Man::Person.
Woman::Person.
```

Unsatisfiable Rule Condition

An unsatisfiable condition exists if for a rule R_1 a literal B_i in the rule body B unifies with neither a fact literal L_j in the knowledge base nor a literal H_k in the rule head H of another rule R_2. This anomaly has been described by [7] and [50].

In the following example the literal *?l:Lecture[teacher −>?x]* does not unify with facts in the knowledge base and there is no rule with a rule head it can unify with. A query for instances of *Teacher* will therefore produce no answer.

```
John:Student.
MathClass1:Lecture.

?x:Teacher :- ?x:Student AND
     (EXIST ?l ?l:Lecture[teacher -> ?x]).
```

Subsumed Rule

According to [7] and [50] if a rule R_1 subsumes a rule R_2 it is redundant. This means that the head H_1 of a rule R_1 subsumes the head H_2 of the rule R_2 and the body B_1 of the rule R_1 subsumes the body B_2 of the rule R_2 but not vice versa. Otherwise they are identical.

In the following example the first rule is subsumed by the second and is therefore redundant.

```
?x:Teacher :-
  ?x:Person AND ?y:Lecture[teacher->?x].

?x:Teacher[course -> ?y] :-
  ?x:Person AND ?y:Lecture[teacher->?x].
```

Redundant Rule

If all answers inferable are in any case the same with and without a rule *R* the rule is redundant. An example for a cause of such a redundancy: Within OntoBroker it is possible to *materialize* [48] rules. When materializing a rule it is evaluated and all inferred facts are added as extensional facts to the knowledge base. In this case the rule is redundant and can be omitted.

In the example below the rule is redundant since the only fact it infers it that *John* is instance of the concept *Teacher*. However this fact is already contained in the fact base.

```
John:Student.
MathClass1:Lecture[teacher->John].
John:Teacher.

?x:Teacher :- ?x:Student AND
     (EXIST ?l ?l:Lecture[teacher->?x]).
```

Redundant Use of Transitivity

A rule *R* of the form:

```
?x[P -> ?y] :- ?x[P -> ?z] AND ?z[P -> ?y].
```

contains a redundant definition of transitivity of a property *P* if *P* is already defined as a transitive property. The anomaly is described in [6].

The following example defines the property *related* as transitive and defines a redundant rule stating the same.

```
Person[related{transitive} *=> Person].

?x[related -> ?z] :- ?x:Person[related -> ?y] AND
     ?y:Person[related -> ?z:Person].
```

Redundant Use of Symmetry

In analogy to the described anomaly above [7] describes the redundant use of symmetry. A rule *R*

```
?x[P -> ?y] :- ?y[P -> ?x].
```

contains a redundant definition of symmetry of a property *P* if *P* is already defined as a symmetric property.

The following example defines the property *married* as symmetric and defines a redundant rule stating the same.

```
Person[related{symmetric} *=> Person].

?x:Person[married -> ?y] :- ?y:Person[married -> ?x].
```

Redundant Use of Inverse Property

In analogy to the described anomaly above the redundant use of inverse can be added. A rule *R*

```
?x[P1 -> ?y] :- ?y[P2 -> ?x].
```

contains a redundant definition of a inverse property *P1* and *P2* if they are already defined as a inverse properties.

The following example defines the property *attends* as inverse of the property *attendants* and defines a redundant rule stating the same.

```
Student[attends{inverseOf(attendants)} *=> Lecture].
Lecture[attendants *=> Student].

?x:Student[Student -> ?y] :- ?y:Lecture[attendants -> ?x].
```

4.1.4 Language Conformance

Undefined Instance Property

This anomaly exists if a property *P* is declared for instances but not in the signature. For reasoning with ObjectLogic this is not an issue. However, it is a good habit to fully define the signature. It has been described in [58].

In the following example the property *attends* is not defined for the concept *Student*.

```
John:Student.
MathClass1:Lecture.
John[attends -> MathClass1].
```

Property Range Type

According to [58] this anomaly exists if the property range of an instance is not conform to its signature.

In the following example the instance *MathClass1* is not in the property's range of the property *attends* because *MathClass1* is not an instance of the concept *Lecture*. The example may be correct, but it is not type safe.

```
Lecture[].
Class[].
Student[attends *=> Lecture].

John:Student.
MathClass1:Class.
John[attends -> MathClass1].
```

Circular Sub-Concepts

Within this anomaly concepts inherit from their own sub-concepts. The anomaly is described within [6].

Within this small example the concept *A* inherits from its own sub-concept *B*.

```
A::B.
B::A.
```

However, the circularity can also be caused within a deeper hierarchy.

```
A::B1.
B1::B2.
B2::A.
```

Having circular sub-concept relation would lead to a concept hierarchy where all circular sub-concepts would have the same properties. Therefore the concepts would be identical.

Circular Properties

Within this anomaly properties inherit from their own sub-properties. This might be directly or within a chain of properties.

```
P1 << P2.
P2 << P1.
```

As with circular sub-concepts the circularity can be caused by a deeper hierarchy of properties as in the following example.

```
P1 << P2.
P2 << P3.
P3 << P1.
```

Circular Dependency (Ambivalent Self-Reference)

The anomaly occurs if a head literal H_i of a rule R is used as body literal B_j in its own body B or if the head of the rule is the only way to derive a not existing fact required in its body. This anomaly is described in [50] and [6].

This kind of circularity occurs when a sub-concept relation exists between two concepts A and B_i

```
A::Bi.
```

and a rule exists that the sub-concept is derived by using the super-concept as a body atom.

```
A :- B1,... , Bn.
```

In the rule of the example below, the conclusion *?x:Teacher* has a circular dependency due to the sub-concept relation between the concept *Teacher* and *Person*.

```
Teacher::Person.

?x:Teacher :- ?x:Person AND ?l:Lecture AND
     ?l[teacher->?x].
```

Due to the fact that *Teacher* is a *Person* the rule head applies for its own body atom *?x:Person* and the rule is called by evaluating the body atom. However, using bottom up evaluation of OntoBroker will not lead to an issue here.

Cardinality Constraint

Within [7] the main focus is set to functional properties. However we think that cardinality in general needs to be checked within ObjectLogic. Within ObjectLogic cardinality is denoted by the cardinality constraints in the form of:

```
Domain[Property {min:max} *=> Range].
```

Where Domain donates the domain of the Property and Range the range of this property. And min is the minimum cardinality and max the maximum cardinality. The cardinality {0:1} enforces the functional property.

The following example contains a constraint on the property *teacher* of the concept *Lecture* that states that a *Lecture* can have only one teacher. However the fact base contains two teachers for the *Lecture MathClass1*.

```
Lecture[teacher {1:1} *=> Person].

John:Person.
Mira:Person.

MathClass1:Lecture[teacher->John].
MathClass1:Lecture[teacher->Mira].
```

4.2 Diagnosis

Section 4.1 introduced issues that might arise when implementing knowledge bases based on ObjectLogic. The following section will address the identification of such issues. Instead of using DATALOG as proposed in [7] we are using ObjectLogic for implementing constraints checking the anomalies. Some of the necessary functionality cannot be provided purely by logic easily. Therefore we introduce some extensions that are used to provide additional functionality for detecting anomalies. Based on the structure of section 4.1, sections 4.2.2, 4.2.3, 4.2.4, and 4.2.5 will introduce different diagnosis options for the different anomalies.

4.2.1 Supporting Functionality

As explained above it is necessary to introduce additionally functionality in order to be able to check the anomalies. At first we introduce some term style representation for ObjectLogic atoms which are used for decomposing rules. This representation lifts the representation of rules on a meta-level in order to make assumptions on rules on meta-level. The reason will become clearer later on. Then we introduce procedural attachments to OntoBroker that are used e.g. for decomposing rules. Some of these attachments are using the term style representation in order to make assumptions on rules on meta-level. Finally we will introduce general rules that are reused by various diagnoses.

Term Style Atoms

In order to make assumptions on rules on a meta-level we need to process them on meta-level. Usually OntoBroker evaluates rules based on the given facts. However, for checking the anomalies we need to check the structure of the rules. Therefore a meta-representation of the ObjectLogic atoms is used by several built-ins (cf. next section).

instanceOf(?a,?b)	?a:?b
subConceptOf(?a,?b)	?a::?b
propertyMember(?a,?b,?c)	?a[?b ->?c]
propertyTransitive(?a,?b)	?a[?b *=>()]
propertyRange(?a,?b,?c,boolean)	?a[?b *=>?c]

Table 4.1: Term Style Representation of ObjectLogic Atoms

The inheritance argument in *propertyRange* is defined as *boolean*. When it is set to *_true* the property is inheritable to sub-concepts. When it is set to *_false* the property is not inheritable.

To embody the combination of terms, the term *multi* is defined. This term can contain multiple terms, for example *multi(instanceOf(a,b),propertyMember(a,p,h))*. The term does not represent a molecule. It just represents a set of atoms.

In order to represent variables in this term style representation we use the representation *_var<No>* where *<No>* is a integer number.

The selected term style representation currently covers only parts of ObjectLogic and could be extended in order to cover e.g. negation and quantifier. However, for a first version we are focusing on some basic ObjectLogic axioms that are frequently used.

OntoBroker Built-ins

The functionality of OntoBroker, can be extended using procedural attachments. The different kinds of procedural attachments supported by OntoBroker are described in [12] and [48]. For checking anomalies we define different built-ins and their intended functionality.

OntoBroker built-ins are procedural attachments that can be used to perform operations that are not well expressible in logics and better formulated by traditional non-declarative means. Built-ins are Java programs that can be seamlessly accessed from ObjectLogic rules and queries via built-in *predicates*. They can perform e.g. arithmetic or string operations. OntoBroker already provides a variety of built-ins. The list of built-ins can be extended by implementing the provided interface. A first starting point on the topic of built-ins within OntoBroker is the OntoBroker user manual [48].

Additionally to built-ins OntoBroker supports aggregates which are special predicates that can reason over sets of data, such as calculating the maximum, the minimum or average values of a set of values. An aggregate can also collect multiple individual values into lists to make them accessible for common further processing. Aggregates are processed differently to all other predicates. Since they collect all bindings to their parameters they are invoked only once per rule. Aggregates must not occur in rule cycles and the tackled values must not occur in the head of rules.

The following built-ins introduced are used for checking the anomalies.

The _ruleContainsPredicate/6 built-in is used in order to check if a rule contains given literals. For representing the literals the term style representation is used as introduced above. This built-in can actually be used in a different way as well where it is used to decompose a rule with a given *ruleID* in order to process the head and body literals of the rule. The signature of the built-in is as following:

```
_ruleContainsPredicate(module,headLiteral,fullHead
     bodyLiteral,fullBody,ruleID)
```

Having the arguments:

module The module to be checked. A *module* can be seen as the ObjectLogic counterpart of an ontology in OWL.

headLiteral The head literals in term style representation of the rule to be checked.

fullHead A boolean value being *_true* if the literals in the argument *headLiteral* are the complete head literals of this rule. It is *_false* if the literals in the argument *headLiteral* are only a subset of the head literals of the rule.

bodyLiteral The body literals in term style representation of the rule to be checked.

fullBody A boolean value being *_true* if the literals in the argument *bodyLiteral* are the complete body literals of this rule. It is *_false* if the literals in the argument *bodyLiteral* are only a subset of the body literals of the rule.

module The id of the rule within the ontology to be checked.

Having the two boolean arguments for head and body literals allows us to use the built-in to check if particular literals are part of the body or head of the rule. The built-in uses the self-defined constant *_null* to ignore the head or body of the particular rule.

The _violatesConstraint/3 built-in is used in order to check if literals represented in term style are violating a certain constraint. The signature of the built-in is as following:

```
_violatesConstraint(module,literals1,literals2,
   constraintID)
```

Having the arguments:

module The module to be checked. A *module* can be seen as the ObjectLogic counterpart of an ontology in OWL.

literals1 The first literals in term style representation to be checked.

literals2 The second literals in term style representation to be checked.

constraintID The id of the constraint to be checked.

The reason of using two lists of literals will get clear e.g. in the diagnosis of *Self-Contradicting Rule* in Section 4.2.2 where head as well as body literals of a rule are checked for violation of constraints.

The _subsumes/2 built-in is used in order to check if a list of literals subsumes another list of literals.

The signature of the built-in is as following:

```
_subsumes(literals1,listerals2)
```

Having the arguments:

literals1 The literals in term style representation as reference.

literals2 The literals in term style representation to be checked against the reference.

The _queryRuleLiteral/3 built-in is used in order to pose the given literal as query to the knowledge base. The signature is:

```
_queryRuleLiteral(module,literal,results)
```

Having the arguments:

module The module to be checked. A *module* can be seen as the ObjectLogic counterpart of an ontology in OWL.

literal The literal to be used for the query in term style representation.

results The query results.

The _ruleContainsTerm/4 built-in is an internal OntoBroker built-in. It is used to find terms in rules. The signature is:

```
_ruleContainsTerm(module,term,isHead,ruleID).
```

Having the arguments:

module The module to be checked. A *module* can be seen as the ObjectLogic counterpart of an ontology in OWL.

term The term that is searched for.

isHead A boolean value stating that the head of the rule is included in the search. This argument has either the constant value *_true* or *_false*.

ruleID The id of the rule within the ontology to be searched in.

The *count* aggregate We are using the *count* aggregate in the diagnosis of *Cardinality Constraint* in Section 4.2.5. The aggregate counts the occurrences of the elements defined in the *query* while the grouping of the counts is done based on the *grouping-vars*.

```
?- ?count = count{ aggr-vars [ grouping-vars ] | query }.
```

Counting persons with a salary larger than 10000 grouped by their department could be expressed by:

```
?- ?N = count{?I [?I, ?D] |
   ?I:Person AND ?I.department == ?D
     AND ?I.salary > 10000}.
```

The query asks for instances of the concept *Person* that work in some department *?D* and have a salary larger than *1000*. Here *I* is the aggregation variable. It is the actual instance of the concept *Person*. The grouping is done via *?I* grouping the results by the individual instances of the concept *Person* as well as *?D* additionally grouping those results by department.

Selecting the ObjectLogic Module

The constraints checking the anomalies as specified below will need to get the *module*-id of the module to be checked. A *module* can be seen as the ObjectLogic counterpart of an ontology in OWL. To make the approach universally applicable we define the rules with variables for the module id. We use a predicate *_cModule/1* to introduce the module id that is actually checked. Having a module id like *module1* one can define the fact *_cModule(module1)* in order to check this module with the proposed approach.

Supporting Rules

This section introduces different rules that are used by different diagnoses for checking the anomalies. Together with the built-ins they build the framework for the checking functionality within ObjectLogic using OntoBroker. The rules are adopted from [7].

The rules calculating the derives property define the relation under what circumstances a concept derives another.

```
?A[derives -> ?B] :- ?A::?B@?m AND
  _cModule(?m).

?A[derives -> ?B] :- _ruleContainsPredicate(
  ?m,instanceOf(_var1,?A),_false,
  instanceOf(_var1,?B),_true,?)
  _cModule(?m).
```

The first rule defines this relation in case a sub-concept relation exists between two concepts *A* and *B*. The sub-concept relation in ObjectLogic is transitive.

The second rule defines the relation *derives* for rules of the form:

```
?a:A :- ?a:B.
```

with a single body literal *?a:B*. Where *A* and *B* are concepts.

The rules deriving the transitive-derives relation define the transitive closure of the *derives* relation.

```
?A[tc_derives -> ?B] :- ?A[derives ->?B].

?A[tc_derives -> ?B] :- ?A[derives -> ?C] AND
  ?C[tc_derives -> ?B].
```

The rule deriving the direct sub-concept relation is introduced for readability and consistency. OntoBroker already comes with an internal predicate for the direct sub-concept relation. However, for readability and consistency we define the relation *directSubConcept*. The later part of the rule *?A::?B@?m* is only needed to determine the module.

```
?A[directSubConcept -> ?B] :- $assertedsub(?A,?B)
  AND ?A::?B@?m
  AND _cModule(?m).
```

The rules deriving the _connectedConceptVia/3 predicate are introduced to calculate the path of a concept which is connected to another concept. In this particular case the connecting relation is the sub-concept relation. The path describes the intermediate concepts via which the two concepts *A* and *B* are connected. The rule creates facts of the predicate *_connectedConceptVia/3* with the first two arguments being the two concepts *A* and *B* that are connected and the third argument being a list *L* of concepts describing the path the two concepts are connected to.

```
_connectedConceptVia(?A,?B,?L) :-
  ?A[directSubConcept -> ?B] AND
  ?A != ?B AND
  ?L = [].

_connectedConceptVia(?A,?B,?L) :-
  ?A[directSubConcept -> ?C] AND
  _connectedConceptVia(?C,?B,?L2) AND
  ?A != ?B AND
  ?A != ?C AND
  ?B != ?C AND
  NOT _contains(?L2,?C) AND
  _concat(?L2,[?C],?L).
```

The rule deriving the direct-sub is introduced for readability and consistency. OntoBroker already comes with an internal predicate for the direct sub-property relation. However, for readability and consistency we define the relation *directSubProperty*. The later part of the rule is only needed to distinguish the module to be checked. The variable *?m* should contain the id of the ontology to be checked. This is analogous to the direct sub-concept definition.

```
?A[directSubProperty -> ?B] :-
  $assertedsubproperty(?A,?B)
  AND ?A << ?B@?m
  AND _cModule(?m).
```

The rules deriving the _connectedPropertyVia/3 predicate are introduced to calculate the path a property is connected to another property via sub-property relation. The path describes the intermediate properties via which the two properties *P1* and *P2* are connected. The rule creates facts of the predicate *connectedPropertyVia/3* with the first two arguments being the two properties *P1* and *P2* that are connected and the third argument being a list *L* of properties describing the path the two properties are connected to. This is analogous to the connected property definition above.

```
_connectedPropertyVia(?P1,?P2,?L) :-
  ?P1[directSubProperty -> ?P2] AND
  ?P1 != ?P2 AND
  ?L = [].

_connectedPropertyVia(?P1,?P2,?L) :-
  ?P1[directSubProperty -> ?P3] AND
  connectedPropertyVia(?P3,?P2,?L2) AND
  ?P1 != ?P2 AND
  ?P1 != ?P3 AND
  ?P2 != ?P3 AND
  NOT _contains(?L2,?P3) AND
  _concat(?L2,[?P3],?L).
```

4.2.2 Contradiction

Ambivalent Rules

A pragmatic way to check this issue is to execute the constraint. The semantic of a constraint is that if the constraints body is executed as query and it returns results it is violated. To check if the violation is due to some rule one can turn inference off with the OntoBroker query option @{*options[inferOff]*}. Turning inference off will only return results already contained within the knowledge base without drawing conclusions firing rules. If the query still returns the same results the query is violated by the fact base. If there is a discrepancy at least some violation is due to the rules. To identify the rule that is violating the constraint one can use e.g. the explanation functionality of OntoBroker [48] to track back the created results. Additionally one can use the ObjectLogic debugger [49] of OntoStudio [1] to track back the causes.

It would be possible to implement a built-in actually doing the different steps automatically. The built-in would need to get the id of the constraint as well as the id of the rule in order to perform. It can fire the constraint as query, one time with and one time without inference off option. It could then compare the result sets.

Ambivalent Rule Pairs (Contradicting Rules)

Contradicting rules are rules that conclude contradicting facts while the body of one of the rules subsumes the body of other rule.

Within ObjectLogic two conclusions are contradicting if they violate a certain given constraint of the form:

```
!- <body literals>.
```

To identify the contradiction of the rule heads we use the previously introduced OntoBroker built-in *_violatesConstraint/4*. The subsumption of the body literals is checked by the OntoBroker built-in *_subsumes/2*. The additional built-in *_ruleContainsPredicate/6* decomposes the rule to its head and body literals. At least one fact for the predicate *_cModule/1* must be defined which defines the module to be checked during runtime.

```
!-
```

[1]OntoStudio: ObjectLogic modeling environment by ontoprise GmbH

```
  _subsumes(?body1,?body2) AND
  _violatesConstraint(?m,?head1,?head2,
    ?constraintID)
  AND
  _ruleContainsPredicate(?m,?head1,_true,?body1,
    _true,?rule1)
  AND
  _ruleContainsPredicate(?m,?head2,_true,?body2,
    _true,?rule2)
  AND
  _cModule(?m).
```

Incompatible Rule Antecedent

It needs to be checked if two or more body literals of a rule are incompatible. This is done by getting the body literals of the rule and checking if they violate a constraint. The functionality is provided by the previously defined OntoBroker built-ins *_violatesConstraint/4* and *_ruleContainsPredicate/6*.

```
!-
  _violatesConstraint(?module,?body,?body,?constraintID)
  AND
  _ruleContainsPredicate(?m,?head,_true,?body,
    _true,?ruleID)
  AND
  _cModule(?m).
```

Self-Contradicting Rule

It needs to be checked if two or more head and body literals of a rule are incompatible. This is done by getting the head and body literals of the rule and checking if they violate a constraint. The functionality is provided by the previously defined OntoBroker built-ins *_violatesConstraint/4* and *_ruleContainsPredicate/6*.

```
!-
  _violatesConstraint(?m,?head,?body,?constraintID)
  AND
  _ruleContainsPredicate(?m,?head,_true,?body,
    _true,?ruleID)
  AND
  _cModule(?m).
```

4.2.3 Completeness

Undefined Concept

In the following constraint we define that there is some violation of the constraint if the instance unified with the variable *?i* is instance of a concept unified with the variable *?A* that is not in some relation with some other concept unified with the variable *?B*, and it is not explicitly declared as concept.

```
!- ?i:?A@?m AND
   (NOT EXIST ?B ?A::?B@?m) AND
   (NOT EXIST ?B ?B::?A@?m) AND
   (NOT EXIST ?P, ?B ?A[?P => ?B]@?m) AND
   (NOT EXIST ?P, ?B ?A[?P *=> ?B]@?m) AND
   (NOT EXIST ?A[]@?m)
  AND
  _cModule(?m).
```

Concept Leaf

In order to have a full check of this anomaly we use the previously introduced OntoBroker built-in *_ruleContainsTerm/4*. It searches for particular terms in the rule. In our case the variable *?A* must be a concept due to the first statement. Therefore we check if there exist rules that contain this term. If not the condition is fulfilled and therefore the constraint violated. Additionally it needs to be checked if the concept has instances or sub-concept relations.

```
!- $concept(?A)@?m
  AND (NOT EXIST ?a ?a:?A@?m)
  AND (NOT EXIST ?B ?B::?A@?m)
  AND (NOT EXIST ?ruleID
     _ruleContainsTerm(?m,?A,_true,?ruleID))
  _cModule(?m).
```

Rule Property Undefined

In order to check if a property used within a rule is defined, one needs to check if the property is used in a rule and if the signature exists in the ontology.

The first step is done by the built-in *_ruleContainsPredicate/6* described in Section 4.2.1. In this case we are particularly looking for properties used within the rule. The first constraint is violated if a property is used within the rule head that is not defined within the schema, the second does the same for the body.

Checking the head properties.

```
!-
   _ruleContainsPredicate(?m,
   multi(propertyMember(_var1, ?P, _var2),
     instanceOf(_var1,?C1),
     instanceOf(_var2,?C2)),_false,
   _null,_true, ?ruleID) AND
  (NOT EXIST ?P, ?C1[?P => ?C2]@?m) AND
  (NOT EXIST ?P, ?C1[?P *=> ?C2]@?m) AND
  _cModule(?m).
```

Checking the body properties.

```
!-
   _ruleContainsPredicate(?m,
   null,_true,
   multi(propertyMember(_var1, ?P, _var2),
     instanceOf(_var1,?C1),
     instanceOf(_var2,?C2)),_false,
     _?ruleID) AND
  (NOT EXIST ?p, ?C1[?P => ?C2]@?m) AND
  (NOT EXIST ?p, ?C1[?P *=> ?C2]@?m) AND
  _cModule(?m).
```

4.2.4 Relevance

Redundancy by Repetitive Taxonomic Definition

The *indirect repetition* can be identified by checking if two concepts are connected by a sub-concept relation. If there exist two different paths of sub-concept relations via other concepts defined in the lists *?L1* and *?L2* the definition of the sub-concept relations is redundant.

```
!-
 _connectedConceptVia(?A,?B,?L1) AND
 _connectedConceptVia(?A,?B,?L2) AND
 ?L1 != ?L2 AND ?A::?B@?m
 AND _cModule(?m).
```

Unsatisfiable Rule Condition

To detect this anomaly one can identify the body literal that is causing the unsatisfiable rule condition by the following way.

- Select a rule.
- Select a body literal of the rule.
- Check if it occurs in a head literal of another rule.
- Check if there are instances that unify with the body literal of the first rule.

The constraint below reflects this approach.

```
!-
   _ruleContainsPredicate(?m,
     null,_true,?literal,_false,
     _?ruleID1) AND
   (NOT EXIST _?ruleID2
     _ruleContainsPredicate(?m,
       ?literal,_false, null,_true,
       _?ruleID2)) AND
   (NOT EXIST _?fact
     _queryRuleLiteral(?m,?literal,?fact))
  _cModule(?m).
```

Subsumed Rule

In Section 4.2.1 we introduced the built-in *_subsumes/2* which checks if a list of literals subsumes another list of literals. This built-in in addition with the rule built-in *_ruleContainsPredicate/6* is used to check the subsumption. The subsumption test is done for the head as well as for the body of the rules.

```
!-
 _ruleContainsPredicate(?m,
   ?head1,_true,?body1,_true,?rule1)
 AND
 _ruleContainsPredicate(?m,
   ?head2,_true,?body2,_true,?rule2)
 AND
 _subsumes(?body1,?body2)
 AND
 _subsumes(?head1,?head2).
 _cModule(?m).
```

Redundant Rule

To check if all conclusions drawn from a rule are already contained within the knowledge base one can query the rule consequent within OntoBroker one time with and one time without inference off using the OntoBroker query option @{*options[inferOff]*}. Turning inference off will only return results already contained within the knowledge base without firing the rules. If the result sets are equal the rule is redundant.

It would be possible to implement a built-in actually doing the different steps automatically. The built-in would need to get the id of the rule in order to perform. It can fire the rule head as query, one time with and one time without inference off option. It could then compare the result sets. If they are equal the built-in is satisfied.

Redundant Use of Transitivity

To check if a rule states a redundant transitivity relation we assume the general form of such definition is:

```
?x[relation -> ?y] :-
   ?x:C[relation -> ?z]
   AND
   ?z:C[relation -> ?y:C].
```

To identify such rules we decompose the rule using the previously introduced built-in *_ruleContainsPredicate/6*. The constraint additionally checks if there is a definition of transitivity for a given property.

```
!-
 _ruleContainsPredicate(?m,
   propertyMember(_var1, ?P, _var3),_false,
   multi(propertyMember(_var1, ?P, _var2),
     propertyMember(_var2, ?P, _var3)),
   _false,?ruleId)
   AND
   (?concept[?P{transitive} *=> ?concept]@?m
     OR
     ?concept[?P{transitive} => ?concept]@?m)
   AND _cModule(?m).
```

Redundant Use of Symmetry

Analogous to the *Redundant Use of Transitivity* we can define a constraint checking the redundant use of symmetry. Assuming that the general form of symmetry is expressed within ObjectLogic in the following form:

```
?x[property -> ?y] :- ?y:C[property -> ?x:C].
```

we define the constraint as following:

```
!-
 _ruleContainsPredicate(?m,
   propertyMember(_var1,?P,_var2),_false,
     propertyMember(_var2,?P, _var1),_false,
     ?ruleId)
   AND
   (?C[?P{symmetric} *=> ?C]@?m
     OR
     ?C[?P{symmetric} => ?C]@?m)
   AND _cModule(?m).
```

The first part decomposes the rule and checks for patterns in the rule as shown in the general example above. The second part checks if there exist signature definitions for the concept and relation that define the relation as symmetric.

Redundant Use of Inverse Property

Analogous to the *Redundant Use of Transitivity* and *Redundant Use of Symmetry* we can define a constraint checking the redundant use for inverse relations. Assuming that the general form of inverse relations is expressed within ObjectLogic in the following form:

```
?x[P1 -> ?x] :- ?y:C1[P2 -> ?x:C2].
```

we define the constraint as following:

```
!-
 _ruleContainsPredicate(?m,
   propertyMember(_var1, ?P1, _var2),_false,
     propertyMember(_var2, ?P2, _var1),_false,
     ?ruleId)
   AND
   (?C1[?P1{inverseOf(?P2)} *=> ?C2]@?m
     OR
     ?C1[?P1{inverseOf(?P2)} => ?C2]@?m)
   AND _cModule(?m).
```

The first part decomposes the rule and checks for patterns in the rule as shown in the general example above. The second part checks if there exist signature definitions for the concept and relation that define the two relations as inverse.

4.2.5 Language Conformance

Undefined Instance Property

The issue can be identified by the following constraint:

```
!-
  ?i1[?P -> ?i2]@?m AND
  ?i1:?C1@?m AND
  ?i2:?C2@?m AND
  (NOT EXIST ?P, ?C2 ?C1[?P => ?C2]@?m) AND
  (NOT EXIST ?P, ?C2 ?C1[?P *=> ?C2]@?m) AND
  AND _cModule(?m).
```

It checks for properties defined on instance level they have a signature definition.

Property Range Type

Checking if the range type of a property is conform to its signature can be done with the following constraint.

```
!-
  ?i1[?P -> ?i2]@?m AND
  ?i1:?C1@?m AND
  (?C1[?P => ?C2]@?m OR
   ?C1[?P *=> ?C2]@?m) AND
   ?i2:?C3@?m AND
   ?C2 != ?C3 AND
   _cModule(?m).
```

It checks for all domain-property-range triple if the range of the property is conform to the signature.

Circular Sub-Concepts

Circular sub-concepts can be detected using the predicate *_connectedConceptVia/3* previously defined in Section 4.2.1. The first application of the predicate checks if a connection exists between a concept *?A* and *?B*. The second application checks if there is a circular definition for the sub-concept relation between the two concepts. The two lists define the path through other concepts that connect the two concepts.

```
!-
_connectedConceptVia(?A,?B,?L1) AND
_connectedConceptVia(?B,?A,?L2) AND
?L1 != ?L2 AND
?A != ?B AND
_cModule(?m).
```

Analogous to [7] one could also include derivations done by rules and not only the subclass derivation. To include this circularity we use the previously defined helper rules defining the properties *derives* and *tc_derives*.

```
!-
   ?A[derives -> ?B]@?m AND
   ?B[tc_derives -> ?A]@?m AND
   ?A != ?B AND
   _cModule(?m).
```

Due to the inclusion of rules for checking derivation we cannot identify the path of the sub-concept hierarchy.

Circular Properties

Circular properties can be detected using the predicate *_connectedPropertyVia/3* previously defined in Section 4.2.1. The first application of the predicate checks if a connection exists between a property *?P1* and *?P2*. The second application checks if there is a circular definition for the sub-property relation of the two properties. The two lists define the path through other properties that connect the two properties.

```
!-
_connectedPropertyVia(?P1,?P2,?L1) AND
_connectedPropertyVia(?P2,?P1,?L2) AND
?L1 != ?L2 AND
?P1 != ?P2 AND
_cModule(?m).
```

Circular Dependency (Ambivalent Self-Reference)

To detect self-references within a rule the rule needs to be decomposed and the individual head and body literals need to be compared. To do so we use the OntoBroker built-in *_ruleContainsPredicate/6*.

```
!-
   _ruleContainsPredicate(?m,?x,_false,?x,_false,?r) AND
   _cModule(?m).
```

The condition checks if there is a particular head literal which is the body literal of the same rule.

It must be stated that this constraint does not cover the case where a concept in the head of a rule is sub-concept of the concept in the rule body as shown in the example of *Circular Dependency* in Section 4.1.4.

The following constraint will cover this particular case.

```
!-
   _ruleContainsPredicate(?m,
     instanceOf(_var1,?C1),_false,
     instanceOf(_var1,?C2),_false,?r)
   AND
   ?C1::?C2 AND
   _cModule(?m).
```

Cardinality Constraint

For checking min and max cardinality we use two different constraints. The OntoBroker aggregate *count* counts the number of property values for each instance. This is stated by the query *?i1[?R → ?i2]* of the aggregate. Then the counted number is checked against the min and max cardinality defined in the signature.

Checking max cardinality:

```
!-
   ?Count = count{?i2 [?i1, ?R] | ?i1[?R -> ?i2]}
   AND ?C1[?P {?:?Max} *=> ?C2] AND ?i1:?C1
   AND ?Count > ?Max AND ?Max > -1.0.
```

Checking min cardinality:

```
!-
   ?Count = count{?i2 [?i1, ?R] | ?i1[?R -> ?i2]}
   AND ?C1[?P {?Max:?} *=> ?C2] AND ?i1:?C1
   AND ?COUNT < ?Min.
```

4.3 Actions

This section describes hints on actions that can be performed in order to solve the issues identified by the methods introduced in Section 4.2. Some of the issues might have more than one action to be applied. In this case the user needs to decide what action he wants to perform.

4.3.1 Contradiction

Ambivalent Rules

Having identified that a rule R contradicts a given constraint C one needs to check if the rule or the constraint itself is incorrect. However, this is a very general anomaly and might have different reasons. In some cases it might be that even a fact in the knowledge base is incorrect. The rule might derive a correct fact but due to the constraint a violation of the constraint is caused. In any case the interaction of the user is needed to solve the issue.

Ambivalent Rule Pairs (Contradicting Rules)

There are different possible causes for the anomaly. In any of these cases the interaction of the user is needed to solve the issue.

One of the rule heads (their conclusion) or even both might be wrong and they can be adopted. In this case one should check the rule heads for their correctness. If one of the heads is incorrect either solve it or check if the rule is needed.

The rule bodies are wrong and the one should not subsume the other. In this case check the rule bodies for their correctness. If one of the bodies should not subsume the other one of the rules need to be adopted.

The constraint is wrong. In this case check the constraint for its correctness. If it is incorrect the constraint needs to be adopted or removed.

Incompatible Rule Antecedent

There are different possible causes for the anomaly. In any of these cases the interaction of the user is needed to solve the issue.

The two body literals B_1 and B_2 of a rule R are incorrectly used together within this rule. In this case one should check if one of the body literals needs to be deleted or adopted.

The constraint is wrong. In this case check the constraint for its correctness. If it is incorrect the constraint needs to be adopted or removed.

Self-Contradicting Rule

There are different possible causes for the anomaly. In any of these cases the interaction of the user is needed to solve the issue.

The body literals B_i or the head literal H_j of a rule R are incorrectly defined together in the rule. In this case one should check if one of the literals needs to be deleted or adopted.

The constraint is wrong. In this case check the constraint for its correctness. If it is incorrect the constraint needs to be adopted or removed.

4.3.2 Completeness

Undefined Concept

In order to solve the anomaly one should define the concept. This can be done by e.g. explicitly stating that the term is a concept *C[].*. Another way would be to identify and implement super-concept *C::SC.* or sub-concepts *SC::C.* or defining properties on the concept *C[P *=>C2].*.

Concept Leaf

The solution of this anomaly is to either remove the concept since it is not used within the knowledge base or it needs to be used within one of the described contexts of *Concept Leaf* in Section 4.1.2. So either

- create an instance of the concept:

  ```
  i:C.
  ```

- define sub-concepts of the concept:

  ```
  S::C.
  ```

- define a rule where it is used as literal in the head or body of the rule:

  ```
  ?x:C :- ?x:B AND ?x[P -> V].
  ```

Rule Property Undefined

If a rule property is undefined it should be defined on schema level:

```
C[P *=> C2].
```

4.3.3 Relevance

Redundancy by Repetitive Taxonomic Definition

One should check if this redundancy is correctly defined or if it can be omitted.

Unsatisfiable Rule Condition

The rule condition is not satisfied and therefore no conclusion can be drawn from the rule. Therefore the rule is dispensable and can be removed from the knowledge base.

Subsumed Rule

Due to the fact that a rule R_1 is subsuming another rule R_2 and therefore deriving all facts rule R_2 is concluding the rule can be removed.

Redundant Rule

The rule does not conclude new facts. Therefore the rule is dispensable and can be removed from the knowledge base.

Redundant Use of Transitivity

The rule does not conclude new facts. Therefore the rule is dispensable and can be removed from the knowledge base.

Redundant Use of Symmetry

The rule does not conclude new facts. Therefore the rule is dispensable and can be removed from the knowledge base.

Redundant Use of Inverse Property

The rule does not conclude new facts. Therefore the rule is dispensable and can be removed from the knowledge base.

4.3.4 Language Conformance

Undefined Instance Property

One should define the property on schema level.

```
C[P *=> C2].
```

Property Range Type

There are different possible causes for the anomaly

1. The range has not been defined correctly and needs to be adopted.
2. The range has been defined correctly but the membership of the instance of the range value is incorrectly defined and needs to be adopted.

Circular Sub-Concepts

The circular sub-concept anomaly can be solved by identifying the wrong sub-concept relation of two concepts and removing it.

Circular Dependency (Ambivalent Self-Reference)

Within bottom-up evaluation of OntoBroker the anomaly is not an issue. However, one should check if the rule is correctly defined leading to this anomaly.

Cardinality Constraint

Cardinality violation can have several causes and therefore different ways to solve the issue.

1. The restriction has been defined incorrectly and need to be adopted.
2. A rule is deriving new facts that lead to the contradiction. In this case it needs to be checked if the rule is correctly defined.
3. The knowledge base already contains facts causing this issue. One should then check if the contained facts are correct.
4. A combination of the last two causes could be the cause. In this case the rule and the facts need to be checked.

Chapter 5

Consistency in Production Rules

The aim of this chapter is to describe some of the consistency problems in business rules and to introduce a new framework for consistency maintenance in a Business Rules systems.

The general idea behind this work is to define a complete path between the symptoms of inconsistency and the solution to these problems, through the identification of the problems corresponding to the symptoms and the choice of the best solutions to these problems. We will track the changes in the components of the rule system (ontologies, business object models, or rules) and detect which changes produce consistency problems and try to provide solutions.

The Changes Managements Pattern is the approach we have developed following this general idea. It is an application of design pattern to rules management, inspired by a similar work applying design patterns to ontology management.

The Chapter is organized as follows, we first describe some consistency problems in production rules, then we introduce CMP our approach for inconsistency diagnosis. Finally we provide a preliminary description of the inconsistency solving method.

5.1 Problems

This section gives a technical description of consistency problems for Production Rules. The description is based on the framework consistency management defined in Task 2.4 and the given examples are inspired from the ArcelorMittal use case.

5.1.1 Contradiction

Definition 1. *A contradiction (or a conflict) is detected in a Knowledge Base KB, if there are at least two rules in the KB, which conduct to a conflicting actions.*

For example, if we define the following rules:

```
r1: IF condition1 THEN action1
r2: IF condition2 THEN action2
```

where $condition_i$ is the condition part of a production rule in $\mathcal{T}(KB)$ and $action_i$ the action part, then the execution of r_1 and r_2 will produce a conflict if $condition_1 \equiv condition_2$ and $action_1$ and $action_2$ are contradictory.

Example 1. *The rules r_1 and r_2 are conflicting because when the yield strength of the coil sampling point is more than the yield strength target of the order of the product, the first one sets true to the assignment while the second make it false.*

```
r1: IF CoilSamplingPoint.yieldStrength > Product.Order.yieldStrengthTarget
    THEN Product.assignment = false
r2: IF CoilSamplingPoint.yieldStrength > Product.Order.yieldStrengthTarget
    THEN Product.assignment = true
```

5.1.2 Irrelevance

The irrelevance consists of five kinds of inconsistencies:

- rules that will never be applied;
- rules that conduct to a domain violation;
- rules that have equivalent condition part;
- equivalent rules;
- redundant rules.

Rule(s) never applied

Definition 2. *A rule never applied inconsistency is detected in KB, if there is at least one rule $r \in KB$ that have a condition part, which is never verified.*

It means that, there is $r \in KB$, for which the condition part will never be true in the KB.

Example 2. *The rule r will never be applied because a mechanical defect cannot be detected for more than 200 meters and for less than 100 meters in the same time.*

```
r: IF there are mechanical defects for
      less than 100 meters
   and there are mechanical defects for
      at least 200 meters
   THEN the phenomenon of the product is a mechanical phenomenon.
```

Domain violation

Definition 3. *A domain violation inconsistency is detected if an action part of a rule sets the value of a property out of the boundary of its domain.*

It means that there is at least one rule $r \in KB$ with an action part that concludes to the modification of the value of $a \in \mathcal{A}(KB)$ and makes it out of range.

Example 3. *r_1 make the value of the yield strength of the coil sampling point out of range because this value cannot be higher than the yield strength upper tolerance of the order of the product.*

```
r1: IF the yield strength of the coil sampling point is less than
       the yield strength lower tolerance of the order of the product
    THEN set the yield strength of the coil sampling point to the yield
       strength upper tolerance of the order of the product + the yield
       strength target of the order of the product.
```

Equivalent condition

Definition 4. *An equivalent condition inconsistency is detected if the KB contains at least two rules that have the same condition part but that their action part are not in conflict.*

It means that there is at least two rules in a rule in the KB that have equivalent condition parts but that produce the same actions or different actions that are not in conflict.

Example 4. *r_1 and r_2 have equivalent condition and they produce the same results when executing them.*

```
r1: IF there are mechanical defects for less than 100 meters or
       there are mechanical defects for at least 200 meters
    THEN the phenomenon of the product is a mechanical phenomenon.

r2: IF there are mechanical defects for less than 100 meters
    THEN the phenomenon of the product is a mechanical phenomenon.
```

Equivalent rules

Definition 5. *An equivalent rules inconsistency is detected if KB contains rules that have the same condition part and the same action part.*

Example 5. *r_1 and r_2 are equivalent because both their conditions and actions are the same. One of them can be removed from the rule set.*

```
r1: IF there are mechanical defects for more than 100 meters
    THEN the phenomenon of the product is a mechanical phenomenon.

r2: IF there are mechanical defects for at least 100 meters
    THEN the phenomenon of the product is a mechanical phenomenon.
```

Redundant rules

Definition 6. *A redundant rules inconsistency is detected if the KB contains rules that make other rules redundant, for instance, if the conditions of one rule are included in the conditions of the other, and the two rules produce the same action.*

Example 6. *r_1 make r_2 redundant because if there are mechanical defects for more than 200 meters there are necessarily mechanical defects for more than 100 meters thus r_2 can be removed from the rule set.*

```
r1: IF there are mechanical defects for more than 100 meters
    THEN the phenomenon of the product is a mechanical phenomenon.

r2: IF there are mechanical defects for more than 200 meters
    THEN the phenomenon of the product is a mechanical phenomenon
```

5.2 Diagnosis

In this section we describe our approach to detect inconsistencies in production rules.

5.2.1 Change Management Pattern

In the deliverable D2.1 [22], we developed the OWL plug-in for JRules that enables authoring business rules grounded in OWL ontologies. OWL ontologies are mapped into the Business Object Model (BOM) of JRules and once the BOM is generated business rules could be authored using the classes and properties of the ontology. Hence, business rules depend on the entities of the ontology and the ontology evolution have impact on the rule set.

In our approach of consistency management, we use design pattern and especially *Change Management Patterns* (CMP). This approach is inspired from ONTO-EVO^{A}L [16] which is used to maintain the consistency of an OWL ontology while it evolves. The CMPs are proposed to guide the evolution process of a rule set while maintaining its consistency.

We define three categories of patterns:

1. *Change Patterns* (CP): classifying types of changes

2. *Inconsistency Patterns* (IP): classifying types of inconsistencies
3. *Repair Patterns* (RP): classifying types of inconsistency resolution alternatives

The rule evolution process is coupled with the ontology evolution process. Hence, the ontology evolution may have impact on the rule set and vice versa. This impact consists of generating consistency problem(s) either on the evolved ontology or the evolved rule set [1]. There is no easy way to establish intuitively the impact of change(s) on the consistency of a rule set, that is why we start by defining the kinds of inconsistencies (see Section 5.1) and the changes that could impact a rule set.

A rule set is composed of rules and each rule has a condition part and an action part. Each condition and action part is composed of the concepts and properties of the domain described by the ontology. We can distinguish two kinds of changes, structural changes and contents changes. Structural changes consist of adding or deleting a rule from the rule set, adding or deleting condition(s) or action(s) of a rule. Content changes consist of changes impacting the concepts and properties used in the rules.

Name	Design the name of the pattern
ID	Design the unique identifier of the pattern
CMP Type	Change Patterns, Inconsistency Patterns or Alternative Patterns
Intent	A description of the goal behind the pattern and the reason for using it
Applicability	Situations in which this pattern is usable; the context for the pattern
Structure	A graphical representation of the pattern. Class diagrams and Interaction diagrams may be used for this purpose
Object	Design the evolved entity (Rule set, or only a rule)
Participants	Gives the ID of the entities impacted by the change
Consequences	A description of the results and side effects caused by using the pattern
Implementation	A description of an implementation of the pattern; the solution part of the pattern
Known Uses	Examples of real usages of the pattern
Related Patterns	Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns
Constraints [2]	Define the constraints that a change should verify before being applicable

Table 5.1: Generic Model of CMP

We define a generic model used to represent the CMPs, the Table 5.1 describes the general properties that every kind of pattern should have. Nevertheless, each type of pattern has

[1]We only focus on consistency management of rule set.

specifics properties as described in:

- Table 5.2 for the pattern of Adding a rule to a ruleset
- Table 5.3 for the pattern of Adding a condition to a rule
- Table 5.4 for the pattern of Adding an action to a rule

Name	Add rule
ID	CP-AddRule-1
CMP Type	Change Patterns
Intent	The pattern models a scenario of adding a rule r in a rule set $\mathcal{R}$ and notifies about the constraints to verify to maintain the coherence of the rule set
Applicability	The pattern is used when adding a rule in a rule set
Scenario	Add the rule r in the rule set $\mathcal{R}$ and notify that there is a rule r' similar to r
Object	$\mathcal{R}$
Participants	r, $\mathcal{R}$
Consequences	r will be added to $\mathcal{R}$ and the consistency of the $\mathcal{R}$ will be maintained
Constraint(s)	$\forall\ r_i \in \mathcal{R}$, C($r$) $\nsubseteq$ C(r_i) $\wedge$ A(r) $\nsubseteq$ A(r_i)
Implementation	A description of an implementation of the pattern; the solution part of the pattern
Known Uses	Examples of real usages of the pattern
Related Patterns	Inconsistency Pattern, Alternative Pattern

Table 5.2: Template of change pattern: Add a rule

Name	Add rule condition
ID	CP-AddRuleCondition-1
CMP Type	Change Patterns
Intent	The pattern models a scenario of adding a condition part $C_i(r)$ to the rule r and notifies about the constraints to verify to maintain the coherence of the rule set
Applicability	The pattern is used when adding a condition part in a rule
Scenario	Add the the condition $C_i(r)$ to the rule r and notify that there is a rule r' similar to r
Object	r
Participants	$r, \mathcal{R}$
Operator	the used operator to add the condition ($\wedge$ or $\vee$)
Consequences	$C_i(r)$ will be added to r and the consistency of the $\mathcal{R}$ will be maintained
Constraint(s)	$\forall\, r_i \in \mathcal{R}$, $C_i(r) \nsubseteq$ C(r_i)
Implementation	A description of an implementation of the pattern; the solution part of the pattern
Known Uses	Examples of real usages of the pattern
Related Patterns	Inconsistency Pattern, Alternative Pattern

Table 5.3: Template of change pattern: Add a rule condition

Name	Add rule action
ID	CP-AddRuleAction-1
CMP Type	Change Patterns
Intent	The pattern models a scenario of adding an action part $A_i(r)$ to the rule r and notifies about the constraints to verify to maintain the coherence of the rule set
Applicability	The pattern is used when adding an action part in a rule
Scenario	Add the the action $A_i(r)$ to the rule r and notify that there is a rule r' similar to r
Object	r
Participants	$r, \mathcal{R}$
Consequences	$A_i(r)$ will be added to r and the consistency of the $\mathcal{R}$ will be maintained
Constraint(s)	$\forall\, r_i \in \mathcal{R}$, $A_i(r) \nsubseteq$ A(r_i)
Implementation	A description of an implementation of the pattern; the solution part of the pattern
Known Uses	Examples of real usages of the pattern
Related Patterns	Inconsistency Pattern, Alternative Pattern

Table 5.4: Template of change pattern: Add a rule action

5.2.2 Example 1: Adding a rule in a rule set $\mathcal{R}$

The constraint to verify when adding a rule r is that there are no rules in the rule set which are similar to this one. Which means that there is no rule that has the same condition or action part.

$$\forall r_i \in R, C(r) \nsubseteq C(r_i) \wedge A(r) \nsubseteq A(r_i)$$

Let us define a rule $r \in \mathcal{R}$, and the rule r' to add to $\mathcal{R}$:

```
r:  IF there are mechanical defects for less than 100 meters or
       there are mechanical defects for at least 200 meters
    THEN the phenomenon of the product is a mechanical phenomenon.

r': IF there are mechanical defects for less than 100 meters
    THEN the phenomenon of the product is a mechanical phenomenon.
```

In this example, the condition parts of r and r' are similar, which implies an equivalent condition inconsistency. To repair this inconsistency r' can be removed and the consistency of the rule set will be maintained.

5.2.3 Example 2: Adding a condition to a rule

Let us define the rule set $\mathcal{R} = \{r, r'\}$:

```
r:  IF there are mechanical defects for more than 100 meters
    THEN set approved to false
r': IF the yield strength of the coil sampling point is more than
       the yield strength target of the order of the product
    THEN set the assignment of the product to true
```

If we consider the change consisting in adding a condition C_2: `there are mechanical defects for less than 150 meters`, with the operator or to r' to produce the following rule:

```
 IF the yield strength of the coil sampling point is more than
    the yield strength target of the order of the product
    or there are mechanical defects for less than 150 meters
 THEN set the assignment of the product to true
```

then r and the new version of r' will produce a conflict when there are mechanical defects for a distance between 100 and 150 meters. To repair the caused inconsistency we have to add another rule that handle the value of the variable the assignment of the product when there are mechanical defects for a distance between 100 and 150 meters.

5.3 Actions

5.3.1 Using CMP to maintain consistency

The proposed approach for consistency maintenance in a production rules ruleset is composed of three steps:

1. Change specification: consist of determining the type of the proposed change (add, update or remove) based on its category (structural or content), the constraints to verify and other properties. The idea here is to make a classification of changes depending on their properties, by defining an ontology of change. Once the change is determined, the corresponding pattern is instantiated.

2. Change analysis: consist of determining and localizing the type of the inconsistency caused by the change based on the type of the change and on the constraint that are not verified.

3. Repair inconsistency: consist of suggesting a solution based on the proposed change and on the inconsistencies caused. It represents additional and/or substitutive changes to implement, to maintain the consistency of the rule set.

5.3.2 Example

Let us define the rule set $\mathcal{R} = \{r_1, r_2, r_3\}$

```
r1: IF the yearly income of the borrower is more than 900
       and the yearly income of the borrower is less than 1200
    THEN set the interest rate of the loan to 0.03%

r2: IF the yearly income of the borrower is more than 1200
       and  the yearly income of the borrower is less than 1500
    THEN set the interest rate of the loan to 0.05%

r3: IF the yearly income of the borrower is more than 1500
    THEN set the interest rate of the loan to 0.07%
```

if we consider the change consisting in adding the rule r' to $\mathcal{R}$

```
r': IF the yearly income of the borrower is less than 1500\\
       or the amount of the loan is more than 2000\\
    THEN set the interest rate of the loan to 0.04\%
```

then we will proceed to the following operations:

1. Change specification: it is the identification of the structural change consisting of add a rule r' to $\mathcal{R}$ so it has to verify this constraint: $\forall\ r_i \in \mathcal{R}$, $\forall\ C_j(r')$, $C_j(r') \nsubseteq$ C(r_i) $\wedge$ A(r) $\nsubseteq$ A(r_i). Then the corresponding pattern is instantiated (see table 5.5).
2. Inconsistency analysis: it is the detection if the inconsistency resulting from executing r' and r_2, it consists of a contradiction on the value of the interest of the loan when the yearly income of the borrower is less than 1500.
3. Resolution alternative: in this case two alternatives could be proposed to resolve the inconsistency problem:
   (a) replace the operator `or` by `and` which will produce a new version of r'

   ```
   r': IF the yearly income of the borrower is less than 1500
          AND the amount of the loan is more than 2000
       THEN set the interest rate of the loan to 0.04%
   ```

   (b) delete the condition C_1: `the yearly income of the borrower is less than 1500`from r', which will produce the new version of r':

   ```
   r': IF the amount of the loan is more than 2000
       THEN set the interest rate of the loan to 0.04%
   ```

Name	Add rule
ID	CP-AddRule-1
CMP Type	Change Patterns
Intent	The pattern models a scenario of adding the rule r' in a rule set $\mathcal{R}$ and notifies that the constraints $\forall\ r_i \in \mathcal{R}$, $C_1(r') \nsubseteq$ C(r_i)$\wedge$ $C_2(r')$ $\nsubseteq$ C(r_i)$\wedge$ A(r')$\nsubseteq$ A(r) must be verified to maintain the coherence of the rule set
Applicability	The pattern is used to add r' in $\mathcal{R}$
Scenario	Add r' in $\mathcal{R}$ and notify that the constraint is not verified
Structure	
Object	$\mathcal{R}$
Participants	r_1, r_2, r_3
Consequences	Detection of entities making the constraint not verified
Constraint(s)	$\forall\ r_i \in \mathcal{R}$, C($r$) $\nsubseteq$ C(r_i) $\wedge$ A(r) $\nsubseteq$ A(r_i)
Implementation	A description of an implementation of the pattern; the solution part of the pattern
Known Uses	Examples of real usages of the pattern
Related Patterns	Inconsistency Pattern, Alternative Pattern

Table 5.5: Change pattern: Add a rule

Chapter 6

Consistency in Description Logics

In this chapter we give a summary of theoretical work on consistency related issues in Description Logics. In the literature the clear focus is on problems of variants of logical inconsistency. Handling logical inconsistency is an important problem field in Description Logics, as it may arise, e.g., due to large amounts of data, like in the context of the Semantic Web, erroneous modelling, ontology integration, and ontology updates. In the next section we give preliminaries on Description Logics. In the subsequent sections, we discuss consistency related problems, diagnosis approaches, respectively actions to address the problems, following the general consistency framework.

6.1 Preliminaries on Description Logics

In the following, we recall syntax and the semantics of Description Logics, using the terminology of Section 2.1. We focus on the basic Description Logic $\mathcal{ALC}$ and will make remarks where language features beyond $\mathcal{ALC}$ are mentioned in the remainder of the document. We assume a signature $\Sigma = (\mathbf{C}, \mathbf{R}, \mathbf{I})$, where $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{I}$ are pairwise disjoint (denumerable) sets of *atomic concepts*, *role names*, and *individual names* respectively.

An $\mathcal{ALC}$ knowledge base $KB = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T} = \mathcal{T}(KB)$ which constitutes the terminological part of the knowledge base and its assertional part $\mathcal{A} = \mathcal{A}(KB)$, called ABox which consists of assertions about actual individuals.

Each atomic concept $A \in \mathbf{C}$, the universal concept $\top$, and the bottom concept $\bot$ are concepts. Moreover, if C and D are concepts and $R \in \mathbf{R}$ is a role name then the following are also concepts:

- the intersection $C \sqcap D$ of C and D,
- the union $C \sqcup D$ of C and D,
- the negation $\neg C$ of C,

- the existential restriction $\forall R.C$ of C by R, and
- the universal restriction $\exists R.C$ of C by R.

A TBox is a finite set of concept inclusion axioms of the form $C \sqsubseteq D$ (meaning the extension of C is a subset of the extension of D; D is more general than C) or $C \equiv D$ (where $C \equiv D$ is interpreted as $C \sqsubseteq D$ and $D \sqsubseteq C$) with C and D being concepts.

An ABox is a finite set of concept assertions of the form $a : C$ where $a \in \mathbf{O}$ and C is a concept, and role assertions of the form $(a, b) : R$, where $a, b \in \mathbf{I}$ are individual names and $R \in \mathbf{R}$ is a role name.

An *interpretation* $I = (\Delta^I, \cdot^I)$ consists of a nonempty *domain* Δ^I and a mapping $\cdot^I$ that assigns to each atomic concept $C \in \mathbf{C}$ a subset of Δ^I, to each individual $o \in \mathbf{I}$ an element of Δ^I, and to each role $R \in \mathbf{R}$, a subset of $\Delta^I \times \Delta^I$. The mapping $\cdot^I$ is defined as follows, where C and D are concepts and $R \in \mathbf{R}$ is a role name:

- $\top^I = \Delta^I$,
- $\bot^I = \emptyset$,
- $(C \sqcap D)^I = C^I \cap D^I$,
- $(C \sqcup D)^I = C^I \cup D^I$,
- $(\neg C)^I = \Delta^I \setminus C^I$,
- $(\exists R.C)^I = \{x \in \Delta^I \mid \exists y \colon (x, y) \in R^I \wedge y \in C^I\}$, and
- $(\forall R.C)^I = \{x \in \Delta^I \mid \forall y \colon (x, y) \in R^I \rightarrow y \in C^I\}$.

A concept inclusion axiom $C \sqsubseteq D$ is satisfied by an interpretation I, symbolically $I \models C \sqsubseteq D$ iff $C^I \subseteq D^I$. Concerning assertional knowledge, a concept assertion $a : C$, respectively a role assertion $(a, b) : R$, is satisfied by I iff $a^I \in C^I$, respectively $(a^I, b^I) \in R^I$.

An interpretation I is a model of a TBox $\mathcal{T}$ iff $I \models t$ for all $t \in \mathcal{T}$. Moreover, I is a model of an ABox $\mathcal{A}$ iff $I \models a$ for all $a \in \mathcal{A}$. Finally, I is a model of an $\mathcal{ALC}$ knowledge base KB iff $I \models \mathcal{T}(KB)$ and $I \models \mathcal{A}(KB)$.

Next we define syntax and semantics of *instance* and *conjunctive queries*. An *instance query* is of the form $A(v)$ and a *conjunctive query* of the form $\exists.\vec{v}.\varphi(\vec{v}, \vec{u})$, where φ is a conjunction of atoms of the form $A(t)$ and $r(t, t')$ where t and t' are individual names or variables. By $\text{terms}(q)$ we denote the set of all individual names and variables in q. Let I be an interpretation and q an instance or conjunctive query with variables $v_1, \ldots, v_k$. For $\vec{a} = a_1, \ldots, a_n \in \mathbf{I}$, an *$\vec{a}$-match* for q in I is a mapping $\pi : \text{terms}(q) \mapsto \Delta^I$ such that $\pi(a) = a^I$ for all $a \in \text{terms}(Q) \cap \mathbf{I}$, $\pi(t) \in A^I$ for all atoms $A(t) \in q$, and $(\pi(t), \pi(t')) \in r^I$ for all atoms $r(t, t') \in q$. Given a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, $\vec{a} = a_1, \ldots, a_k \in \mathbf{I}$ is a *certain answer* to q w.r.t. $\mathcal{T}$ and $\mathcal{A}$ if there is an $\vec{a}$-match for all models I of $\mathcal{T}$ and $\mathcal{A}$.

6.2 Problems

6.2.1 Contradiction and Relevance

Classical *inconsistency* of a Description Logics knowledge base KB is given when it has no model, i.e., there is no interpretation I such that both, $I \models \mathcal{A}(KB)$ and $I \models \mathcal{T}(KB)$.

There are different forms of terminological contradiction in Description Logic TBoxes. On the one hand, there is classical inconsistency, i.e., a TBox is *inconsistent* if it has no model. On the other hand, knowledge engineers are often interested in more fine-grained aspects of satisfiability, namely satisfiability of individual concepts. In terms of our general framework these belong to both types of problems, *contradiction* and *relevance*, since unsatisfiable concepts can be seen as irrelevant.

Definition 7. *A concept C is* unsatisfiable *w.r.t. a TBox $\mathcal{T}$, respectively a Description Logic knowledge base KB, if $C^I = \emptyset$ for all models I of $\mathcal{T}$, respectively for all models I of $\mathcal{T}(KB)$.*

A TBox $\mathcal{T}$, respectively a Description Logic knowledge base KB, is incoherent *if there is an atomic concept in $\mathcal{T}$, respectively in $\mathcal{T}(KB)$, that is unsatisfiable.*

Besides these notions of contradictions, one can also have a look at problems in the presence of queries to a DL knowledge base. Here, the notions of query and predicate emptiness are important [4]. These properties show indicate whether a designed query can produce a non-empty answer and whether a given concept name can be used in a query in a meaningful way.

Query-emptiness is defined as follows.

Definition 8. *Let Σ be a signature and $\mathcal{T}$ a TBox. A conjunctive or instance query q is* empty *for Σ given $\mathcal{T}$ if for all Σ ABoxes $\mathcal{A}$ that are consistent w.r.t. $\mathcal{T}$, it holds that the set of all certain answers to q w.r.t. $\mathcal{A}$ and $\mathcal{T}$ is empty.*

Definition 9. $\mathcal{IQ}$-query emptiness*, respectively* $\mathcal{CQ}$-query emptiness*, is the problem of deciding, given a TBox $\mathcal{T}$, a signature Σ, and an instance query q, respectively a conjunctive query q, whether Q is empty for Σ given $\mathcal{T}$.*

Next, we define predicate-emptiness.

Definition 10. *Let Σ be a signature and $\mathcal{T}$ a TBox. A concept name (predicate) P is* $\mathcal{IQ}$-empty *for Σ given $\mathcal{T}$ if all instance queries q where P occurs in q are empty for Σ given $\mathcal{T}$. Likewise, P is* $\mathcal{CQ}$-empty *for Σ given $\mathcal{T}$ if all conjunctive queries q where P occurs in q are empty for Σ given $\mathcal{T}$.*

Definition 11. $\mathcal{IQ}$-predicate emptiness*, respectively* $\mathcal{CQ}$-predicate emptiness*, is the problem of deciding, given a TBox $\mathcal{T}$, a signature Σ, and an atom concept P whether P is $\mathcal{IQ}$-empty, respectively $\mathcal{CQ}$-empty, for Σ given $\mathcal{T}$.*

6.2.2 Language Conformance

There are many different Description Logics with varying language constructs. Also in standards like OWL 2 [44], different language fragments are distinguished. Hence, determining in which language fragment a given knowledge base falls can be an important issues. As the underlying logic gives bounds on the complexity of reasoning and determines which Description Logics reasoner can be used, knowing the corresponding Description Logic is a requirement in applications. Tools for classifying Ontologies were introduced in [37].

6.3 Diagnosis

6.3.1 Explanation

Generally, explanations are information that can help humans to understand a given set of facts. Here, we are interested in explanations for semantic properties of Description Logic knowledge bases. Such information can be used in two ways. On the one hand, it helps to understand the consequences of a knowledge base and their interrelations, and on the other hand, it can be used to detect and locate problems of different kinds in the knowledge base. Often potential repairs for a problem can be immediately obtained from adequate explanations. In most of the previous work, explanations in Description Logics are given in a proof-theoretic presentation.

McGuiness and Borgida [43] considered the issue of *explaining* subsumption reasoning in Description Logics. Concept subsumption is considered as a main reasoning task in Description Logics.

Definition 12. *Let $\mathcal{T}$ be a TBox. A concept C is subsumed by a concept D with respect to $\mathcal{T}$, symbolically $\mathcal{T} \models C \sqsubseteq D$, if $C^I \subseteq D^I$ for every model I of $\mathcal{T}$.*

Concept subsumption is relevant for different consistency problems in Description Logics.

- Incoherence: Subsumption checking can be used to identify *incoherent concepts*, i.e., concepts that can be proven to have no satisfying instances. In particular, a concept C is incoherent in a TBox $\mathcal{T}$ if $\mathcal{T} \models C \sqsubseteq \bot$. Regarding the classification of problems outlined in the consistency framework in Chapter 2, incoherent concepts can generally be considered as *irrelevant*. If such a concept is used in a meaningful way within the knowledge base it can replaced by the bottom concept and can therefore also be considered as *redundant*. In many cases however, having an incoherent concept C in a DL knowledge base KB is unintended and can therefore easily lead to *contradiction* related problems. E.g.,
  - we have an *assertional contradiction* if some individual is asserted to be contained in C.

  - Moreover, a *terminological contradiction* occurs if KB contains an axiom $D \sqsubseteq C$, where concept D is guaranteed to contain some individual.

- Equivalence: By checking whether two concepts C and D mutually subsume each other it can be checked whether they are equivalent, i.e., contain the same individuals in each model. In such a case C and D can be regarded as *redundant*.

Explanation techniques for concept subsumption can therefore be used for analyzing relevance and contradiction problems as the obtained explanations can guide the knowledge engineer to the source of the problem in the knowledge base.

McGuiness and Borgida [43] propose to explain concept subsumption in the CLASSIC system [10] by natural deduction style proofs. To this end they defined a number of natural deduction rules for deriving concept subsumptions. For avoiding lengthy proofs the authors propose some filtering techniques as well as a two phase search strategy. For explaining a given subsumption $C \sqsubseteq D$ they propose to first find conjunctions $A_1 \sqcap \cdots \sqcap A_n$ of concepts being equivalent to D such that $A_1, \ldots, A_n$ fulfill certain simplicity properties. Then, it has to be shown that $C \sqsubseteq A_i$ for all $1 \leq i \leq n$. Here an aim is to choose the individual A_i, called *atomic descriptions*, in a way such that they do not contain further conjuncts themselves in order to prefer breadth over depth of search. An explanation system could then in the first step return a list of atomic descriptions from which the user can select interesting ones for further explanation. The authors show that this search strategy however does not work for more expressive Description Logics involving role compositions. Moreover, the approach relies on the applicability of structural subsumption algorithms for the considered language, i.e., subsumption reasoning can be done by normalizing descriptions and subsequent syntactic comparisons. However, structural subsumption does not work for more expressive Description Logics. Reasoning in such Description Logics is mainly done by tableaux based algorithms. Unfortunately, such systems are rather unsuitable for explanations as they are based on refutation. Thus, a subsumption $C \sqsubseteq D$ would be proven by unsatisfiability of $C \sqcap \neg D$ which might not be a sufficient explanation for the inexperienced user.

Borgida et al. [9] choose another proof-theoretic approach for explaining subsumption in $\mathcal{ALC}$. Their method is based on a Gentzen-style calculus [27]. They developed sequent rules that are parallel to the rules used in a tableaux system and tailored to be easy to understand. In particular, rules are avoided that move formulas from sequent antecedents to sequent succedents and vice versa, as they are considered potentially confusing by the authors. Moreover, the inference rules are designed such that the structure of the two concepts at hand is maintained. The correspondence to tableaux rules is beneficial as explanations can then be obtained by standard reasoning algorithms using a tagging method sketched by the authors together with lazy unfolding [31].

A third proof-theoretic approach for explanation is based on resolution [14, 13]. Here, the aim is not to explain subsumption but unsatisfiability and inconsistency queries w.r.t. $\mathcal{ALC}$ TBoxes and ABoxes. If a concept is unsatisfiable or an ABox or TBox is inconsistent, they are first translated to first-order logic. Then, a resolution based automated theorem prover is used to generate a resolution proof. This proof is then translated to a corresponding *refutation graph* [17]. This is a graph whose nodes are literals grouped together to clauses such that the edges connect complementary literals within the clauses corresponding to the steps of the resolution proof. Explanations are then generated as a list obtained from a traversal of the refutation graph, where for each literal node passed its associated source axioms are entered. The authors propose to translate this list into a natural language representation.

6.3.2 Pinpointing

In contrast to explanation, where the focus is on allowing humans to understand the outcome of a reasoning process, one can also be interested in the *root cause* of a problem only. That is, instead of clarifying why something went wrong, the aim is to *pinpoint* artifacts, such as conflicting axioms, that are responsible for some undesired property of the knowledge base, such as incoherence of the TBox. The motivation for that it to identify parts of the knowledge base that need to be fixed.

Finding the core reasons for local incoherences in TBoxes was first addressed by Schlobach and Cornet [57]. They introduced the notions of minimal unsatisfiability-preserving sub-TBoxes (MUPS).

Definition 13. *Let C be a concept that is* unsatisfiable *w.r.t. a TBox $\mathcal{T}$. A set $\mathcal{T}' \subseteq \mathcal{T}$ is a MUPS for C in $\mathcal{T}$ if C is unsatisfiable in $\mathcal{T}'$ and C is satisfiable in every $\mathcal{T}'' \subset \mathcal{T}'$.*

Moreover, they defined minimal incoherence-preserving sub-TBoxes (MIPS).

Definition 14. *A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a MIPS of $\mathcal{T}$ if $\mathcal{T}'$ is incoherent and every $\mathcal{T}'' \subset \mathcal{T}'$ is coherent.*

The authors devised a specialized algorithm for computing MIPS and MUPS in $\mathcal{ALC}$ TBoxes that are unfoldable. A TBox is unfoldable when the left-hand side of the axioms are atomic and the right-hand sides contain no reference to the defined concept [45]. The algorithm is based on Boolean minimization of axioms in a tableaux proof. A similar algorithm has been introduced earlier by Baader and Hollunder [3] that has later been extended to $\mathcal{ALC}$ TBoxes with general concept inclusions that are not required to be unfoldable [39].

MIPS and MUPS can be seen as minimal sets of components of the terminology that need to be fixed or removed to restore satisfiability or coherence, following the ideas of model-based diagnosis [54]. Based on Reiter's Hitting Set algorithm, methods for computing MIPS and MUPS where considered [56, 25].

Schlobach and Cornet also introduced the term *axiom pinpointing* for computing MIPS and MUPS which was subsequently used by many authors. Their work has later been extended to $\mathcal{SHIF}$, the Description Logic underlying OWL-Lite [36]. Moreover, a general approach for deriving pinpointing algorithms from tableaux algorithms has been investigated by Baader and Peñaloza [5]. Here, a main research question for which Description Logics techniques like in [57, 3] can be applied easily. For certain types of axioms such as transitive roles or general concept inclusions, tableaux algorithms require certain blocking conditions that need special treatment (like the algorithm in [39]) for ensuring termination. One finding in [5] was that termination of a pinpointing algorithm obtained from a tableaux without major modifications is non-trivial. In particular it is not implied in general by the termination of the tableaux, even if all tableaux rules are deterministic. The authors show that termination is guaranteed when the tableaux produces forest-like structures in their framework.

An approach towards pinpointing that uses both modified tableaux algorithms and Reiter's Hitting Set Tree (HST) algorithm [54] is given in the thesis of Kalyanpur [35]. In this work, the problem of finding all MUPS of an unsatisfiable concept is reduced to finding all *justifications* for a given entailment.

Definition 15. *Let KB be a knowledge base and α a sentence such that $KB \models \alpha$. A fragment $KB' \subseteq KB$ is a* justification *for α in KB if $KB' \models \alpha$, and $KB'' \not\models \alpha$ for every $KB'' \subset KB'$.*

Since justifications for an unsatisfiable entailment can be seen as minimal conflict sets in Reiter's theory, and by computing minimal hitting sets all minimal conflict sets can be obtained, HST can be used to obtain all justifications for a given entailment. First, a justification KB_R for the negation of α is computed using the modified tableaux approach [35]. KB_R serves as root note for the hitting set tree. Subsequently, new leaf nodes are added by removing an arbitrary axiom A from the justification KB of a former leaf node such that the edge is labelled with A. The algorithm then checks consistency with respect to $KB \setminus \{A\}$. If it is inconsistent, then we obtain another justification for α w.r.t. $K \setminus \{i\}$. The algorithm repeats this process, namely removing an axiom, adding a node, checking consistency and performing axiom tracing until the consistency test turns positive.

6.3.3 Checking Query and Predicate Emptiness

Baader et al. [4] investigated query and predicate emptiness, as defined in Section 6.2. Altough considered as a general reasoning task, the motivation for reasoning services based on these notions is detecting whether there is a chance that a designed query can produce a non-empty answer and whether a given concept name can be used in a query in a meaningful way. The authors distinguish between *fixed* and *free queries*. A fixed query is formulated in advance where only the signature of the ontology has to be known. In such a scenario checking query-emptiness can be used for detecting problems with a given query. In the free query setting, where concrete queries are formulated later and not known yet, it makes sense to check for predicate emptiness, i.e., to detect whether a given concept name can be potentially be useful in some query. Baader et al. studied decidability

DL	$\mathcal{IQ}$-query	$\mathcal{CQ}$-predicate	$\mathcal{CQ}$-query
$\mathcal{EL}$	in PTime	in PTime	in PTime
$\mathcal{EL}_{\bot}$	ExpTime-complete	in ExpTime-complete	in 2-ExpTime, ExpTime-hard
$\mathcal{ELI}$	ExpTime-complete	in ExpTime-complete	
$DL\text{-}Lite_{\mathrm{core}}$	in PTime	in PTime	co-NP-complete
$DL\text{-}Lite_{\mathrm{horn}}$	co-NP-complete	co-NP-complete	co-NP-complete
$\mathcal{ALC}$	in NExpTime, ExpTime-hard	in NExpTime, in ExpTime-hard	in 2-ExpTime, ExpTime-hard
$\mathcal{ALCI}$			2-ExpTime-complete
$\mathcal{ALCF}$	undecidable	undecidable	undecidable

Figure 6.1: Complexity Results for Query- and Predicate-Emptiness by Baader et al. [4]

and complexity of checking query and predicate emptiness for several Description Logics. Their results are summarized in Figure 6.3.3.

6.4 Actions

About inconsistency handling of ontologies based on description logics, two fundamentally different approaches can be distinguished. The first is based on the assumption that inconsistencies indicate erroneous data which is to be repaired in order to obtain a consistent knowledge base, e.g., by selecting consistent subsets for the reasoning process [32], or revising the original ontologies with new contradict information [53]. In this respect also pinpointing, as described in the previous section, can be seen as *action* towards handling contradiction when fixing the parts of the knowledge base that where identified as problematic. The other approach yields to the insight that inconsistencies are a natural phenomenon in realistic data which are to be handled by a logic which tolerates it. Such logics are called *paraconsistent*, and the most prominent of them are based on the use of additional truth values standing for underdefined (i.e., neither true nor false) and overdefined (or contradictory, i.e., both true and false). Such logics are appropriately called four-valued logics. We believe that either of the approaches is useful, depending on the application scenario.

6.4.1 Repair Based Approaches

Selection Function

Huang et al. developed a general framework for handling inconsistency in ontologies based on selection function [32]. A selection function is used to determine which consistent subsets of an inconsistent ontology should be considered in its reasoning process.

The general framework is independent of the particular choice of selection function. The selection function can either be based on a syntactic approach , or based on semantic relevance like for example in computational linguistics as in Wordnet.

Revisions in Description Logics

Revision of a Description Logic-based ontology to incorporate newly received information consistently is an important problem for the lifecycle of ontologies [52]. Many approaches in the theory of belief revision have been applied to deal with this problem and most of them focus on the postulate or logical properties of a revision operator in Description Logics [53, 52].

A revision operator in Description Logics is an operation that maps an ordered pair of DL knowledge bases to a set of ontologies such that each of the revised ontology should be consistent and can infer every axiom in the second ontology, and that can be constrained by a set of postulates.

6.4.2 Inconsistency Tolerant Reasoning

Four Valued Description Logics

Instead of the two truth values (true and false) used in classical logic, four truth values uses four truth values, namely *True, False, Both, and None* [8]. By using *Both* for contradict information and *None* for incompleteness, 4-valued logic tolerates inconsistency. Ma et al. introduced the 4-valued semantics to Description Logics for handling the inconsistency in DL based ontologies [42, 41].

The algorithm for 4-valued Description Logics can be classified as two categories. The first approach is directly designing reasoning algorithm, e.g., paraconsistent resolution for $\mathcal{SHIQ}4$ [42]. However, such algorithms mean implementing the reasoner from scratch, which leads to a lot of work. The second one is reducing the reasoning under 4-valued semantics to reasoning under classical semantics. Thus we can use state-of-the-art DL reasoners, such as Pellet, to do the paraconsistent reasoning over inconsistent ontologies.

One drawback of 4-valued Description Logics is the three inclusion semantics, namely material inclusion, internal inclusion and strong inclusion, used for capturing the different parts of the classical semantics for the inclusion. Then the users have to choose the appropriate inclusion for each TBox assertion themselves. Another drawback is the weakness of the reasoning ability. For example, resolution laws does not hold for 4-valued inference relation ($\models_4$): $\{(C \sqcup D)(a), \neg D(a)\} \not\models_4 C(a)$.

Quasi Classical Description Logics

To improve the reasoning of 4-valued Description Logics, Zhang et al. introduced Quasi Classical (QC) semantics to Description Logic [59]. In QC logic, two semantics, namely the strong and weak semantics are used. The weak semantics is the same with 4-valued semantics, which is used for paraconsistent reasoning, while the strong semantics is used to enhance the reasoning ability, e.g., resolution laws are satisfied in QC inference relation ($\models_Q$): $\{(C \sqcup D)(a), \neg D(a)\} \models_Q C(a)$.

Reasoning algorithms for QC Description Logics can also be classified as two categories. One is directly algorithm, such as QC tableau; the other one is reducing the QC reasoning to classical reasoning.

Chapter 7

Consistency in Combinations

In this chapter we present initial results on consistency maintenance in combinations of Description Logics and logical rules. As in Deliverable 3.2 [30], we deal with loosely-coupled combinations, in particular we focus on DL-programs [18]. Based on the preliminaries on Description Logics in Section 6.1, we next outline the theoretical background of DL-programs.

7.1 Preliminaries on DL-Programs

Syntax

A signature $\Sigma = \langle \mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{I} \rangle$ for DL-programs consists of a set $\mathbf{I}$ of 0-ary function symbols and sets $\mathcal{P}_o$, $\mathbf{P}$ of predicate symbols such that $\Sigma_o = \langle \mathbf{C}, \mathbf{R}, \mathbf{I} \rangle$ is a DL-signature and $\Sigma_p = \langle \mathbf{P}, \mathbf{I} \rangle$ is an LP-signature.

Informally, a DL-program consists of a Description Logic ontology Φ over Σ_o and a normal logic program Π over Σ_p, which may contain queries to Φ. Roughly, in such a query, it is asked whether a certain Description Logic formula or its negation logically follows from Φ or not.

A *DL-atom* $a(\mathbf{t})$ has the form

$$\mathrm{DL}[S_1\ op_1\, p_1, \ldots, S_m\ op_m\, p_m;\ Q](\mathbf{t})\,, \qquad m \geq 0, \tag{7.1}$$

where each S_i is either a concept from $\mathbf{C}$ or a role predicate from $\mathbf{R}$, $op_i \in \{\uplus, \cup\!\!\!\!-, \cap\!\!\!\!-\}$, p_i is a unary, resp. binary, predicate symbol from $\mathbf{P}$, and $Q(\mathbf{t})$ is a DL-query. We call $\gamma = S_1\ op_1\, p_1, \ldots, S_m\ op_m\, p_m$ the *input signature* and $p_1, \ldots, p_m$ the *input predicate symbols* of $a(\mathbf{t})$. Moreover, literals over input predicate symbols are *input literals*. Intuitively, $op_i = \uplus$ (resp., $op_i = \cup\!\!\!\!-$) increases S_i (resp., $\neg S_i$) by the extension of p_i, while $op_i = \cap\!\!\!\!-$ constrains S_i to p_i. A *DL-rule* r has the form

$$a \leftarrow b_1, \ldots, b_k, not\ \ b_{k+1}, \ldots, not\ \ b_m\,, \qquad m \geq k \geq 0\,, \tag{7.2}$$

where a is a classical literals and any literal $b_1, \ldots, b_m \in B(r)$ may be a classical literal or a DL-atom. A *DL-program* $\mathcal{KB} = (\Phi, \Pi)$ consists of a DL ontology Φ and a finite set of DL-rules Π.

Semantics

In the sequel, let $\mathcal{KB} = (\Phi, \Pi)$ be a DL-program over $\Sigma = \langle \mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{I} \rangle$. By $gr(\Pi)$ we denote the grounding of Π w.r.t $\mathbf{I}$, i.e., the set of ground rules originating from DL-rules in Π by replacing, per DL-rule, each variable by each possible combination of constants in $\mathbf{I}$.

An *interpretation* I (over Σ_p) is a consistent subset of literals over Σ_p. We say that I satisfies a classical literal l under Φ, denoted $I \models^{\Phi} l$, iff $l \in I$, and a ground DL-atom $a = DL[S_1 \, op_1 \, p_1, \ldots, S_m \, op_m p_m; Q](\mathbf{c})$ under Φ, denoted $I \models^{\Phi} a$, if $\Phi \cup \tau^I(a) \models Q(\mathbf{c})$, where the extension $\tau^I(a)$ of a under I is defined as $\tau^I(a) = \bigcup_{i=1}^{m} A_i(I)$ such that

- $A_i(I) = \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \uplus$;
- $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}$, for $op_i = \mathbin{\cup\!\!\!\!-}$;
- $A_i(I) = \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \notin I\}$, for $op_i = \mathbin{\cap\!\!\!\!-}$.

We say that I *satisfies* the positive (resp., negative) body of a ground DL-rule r under Φ, symbolically $I \models^{\Phi} B(r)^+$ (resp., $I \models^{\Phi} B(r)^-$), if $I \models^{\Phi} l$ (resp., $I \not\models^{\Phi} l$) for all $l \in B(r)^+$ (resp., $l \in B(r)^-$). I *satisfies* the body of r under Φ, denoted $I \models^{\Phi} B(r)$, whenever $I \models^{\Phi} B(r)^+$ and $I \models^{\Phi} B(r)^-$. I satisfies a ground DL-rule r under Φ, symbolically $I \models^{\Phi} r$, if $I \models^{\Phi} H(r)$ whenever $I \models^{\Phi} B(r)$. I is a model of a DL-program $\mathcal{KB} = (\Phi, \Pi)$, denoted $I \models \mathcal{KB}$, iff $I \models^{\Phi} r$ for all $r \in gr(\Pi)$. We say $\mathcal{KB}$ is *satisfiable* (resp., *unsatisfiable*) iff it has some (resp., no) model.

In what follows, we base the answer set semantics of DL-programs on the Faber-Leone-Pfeifer reduct [24].

Definition 16. *Let $\Sigma = \langle \mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{I} \rangle$ be a signature for DL-programs, Φ a DL knowledge base over $\langle \mathbf{C}, \mathbf{R}, \mathbf{I} \rangle$, Π a set of ground DL-rules over $\Sigma_p = \langle \mathbf{P}, \mathbf{I} \rangle$, and I an interpretation over Σ_p. The* FLP-reduct $\Pi^{I,\Phi}_{FLP}$ *of Π under Φ relative to I is the set of rules $r \in \Pi$ such that $I \models^{\Phi} B(r)$. Moreover, the* FLP-reduct $\mathcal{KB}^I_{FLP}$ *of a (possibly non-ground) DL-program $\mathcal{KB} = (\Phi, \Pi)$ relative to I is given by $gr(\Pi)^{I,\Phi}_{FLP}$.*

Definition 17. *Let $\mathcal{KB}$ be a DL-program over $\Sigma = \langle \mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{I} \rangle$. An interpretation I over Σ_p is an* answer set *of $\mathcal{KB}$ if it is a minimal model of $\mathcal{KB}^I_{FLP}$. The set of all answer sets of $\mathcal{KB}$ is denoted by* AS($\mathcal{KB}$).

We use this answer set semantics (we will sometimes refer to it as *FLP-semantics*) rather than one based on the traditional Gelfond-Lifschitz reduct [26], as it naturally handles DL-atoms which are not *monotonic*.

Definition 18. *For a DL-program $\mathcal{KB} = (\Phi, \Pi)$, a ground DL-atom l is* monotonic *relative to $\mathcal{KB}$, if for all interpretations I, J with $I \subseteq J$, $I \models^{\Phi} l$ implies $J \models^{\Phi} l$. $\mathcal{KB}$ is* monotonic *if $gr(\Pi)$ contains only DL-atoms that are monotonic relative to $\mathcal{KB}$.*

It was shown in [20] that for DL-programs that do not employ the $\cap\!\!\!\!-$ operator, the FLP-semantics coincides with strong answer set semantics, as originally introduced for DL-programs [18] using the Gelfond-Lifschitz reduct. Note that this operator is rarely used in practice and can in many cases be removed by simple translations.

We will later refer to the class of *positive DL-programs*, defined in [18] as follows.

Definition 19. *A DL-program $\mathcal{KB}$ is* positive*, if it is monotonic and $B(r)^{-} = \emptyset$ for each rule $r \in \Pi$.*

Note that a DL-atom a that does not employ the operator $\cap\!\!\!\!-$ is always monotonic as $I \subseteq J$ implies $\tau^{I}(a) \subseteq \tau^{J}(a)$.

7.2 Problems

7.2.1 Contradiction

The semantics of a DL-program KB is given by its answer sets $\mathrm{AS}(KB)$. We call KB inconsistent if it has no answer sets, i.e., $\mathrm{AS}(KB) = \emptyset$. In some applications of DL-programs the absence of answer sets can be desired, e.g., when the encodings are designed such that answer sets correspond to unwanted behavior of a system. However, in many cases an inconsistent DL-program is not intended. Then, it is often hard to detect reasons for that. As we are in particular interested in contradiction that is rooted in the combination of ontology and rules, in Section 7.3 we will discuss how to identify minimal sets of calls from the rule part to the DL-part of a DL-program.

7.2.2 Language Conformance: Inconsistency when Combining Ontologies and Rules

The flow of information from the rules to the ontology provides a powerful tool, as results from the program can be used as assertions in the DL knowledge base for further deduction. However, it is possible that the assertions by which the ontology is extended cause an inconsistency in the sense of logical contradiction. We say that in such a case the respective DL-atom is *DL-inconsistent*. As then the respective query is trivially true, we may end up with counterintuitive results, even though both the DL and the LP are perfectly consistent in separation. Hence, this problem is not a matter of contradiction per se, but falls into the language conformance category of the problem framework. Note that DL-inconsistency is not in general an unwanted effect as it is also exploited in some applications of DL-programs, e.g., when using DL-programs for default reasoning [18].

Example 7 (Product Database). *As a running example, we will adapt an example that has been used previously in the context of DL-programs [21].*

A small computer store obtains its hardware from several vendors. It uses the following DL knowledge base Φ_{ex}, which contains information about the product range that is provided by each vendor. For some parts, a shop may already be contracted as supplier and shops which are known to be disapproved for some reason can never become an actual supplier.

$$\begin{aligned}
&\geq 1\ \mathit{supplier} \sqsubseteq \mathit{Shop};\ \ \top \sqsubseteq \forall \mathit{supplier}.\mathit{Part};\\
&\exists \mathit{supplier}.\top \sqcap \mathit{disapproved} \sqsubseteq \bot;\\
&\mathit{Shop}(s_1);\ \ \mathit{Shop}(s_2);\ \ \mathit{Shop}(s_3);\ \ \mathit{disapproved}(s_2);\\
&\mathit{Part}(\mathit{harddisk});\ \ \mathit{Part}(\mathit{cpu});\ \ \mathit{Part}(\mathit{case});\\
&\mathit{provides}(s_1, \mathit{cpu});\ \ \mathit{provides}(s_1, \mathit{case});\ \ \mathit{provides}(s_2, \mathit{cpu});\\
&\mathit{provides}(s_3, \mathit{harddisk});\ \ \mathit{provides}(s_3, \mathit{case});\\
&\mathit{supplier}(s_3, \mathit{case});
\end{aligned}$$

Here, the first two axioms determine Shop and Part as domain and range of the property $\mathit{supplier}$, respectively, while the third axiom constitutes the incongruity between shops that are contracted as supplier but are explicitly disapproved.

Consider the DL-program $\mathcal{KB}_{ex} = (\Phi_{ex}, \Pi_{ex})$, with Π_{ex} given as follows, choosing not-deterministically a vendor for each needed part:

(1) $\mathit{needed}(\mathit{cpu});\ \ \mathit{needed}(\mathit{harddisk});\ \ \mathit{needed}(\mathit{case});$
(2) $\mathit{alreadyContracted}(P) \leftarrow \mathrm{DL}[;\ \mathit{supplier}](S, P), \mathit{needed}(P);$
(3) $\mathit{offer}(S, P) \leftarrow \mathrm{DL}[;\ \mathit{provides}](S, P), \mathit{needed}(P), \mathit{not}\ \mathit{alreadyContracted}(P);$
(4) $\mathit{chosen}(S, P) \leftarrow \mathit{offer}(S, P), \mathit{not}\ \ \mathit{notChosen}(S, P);$
(5) $\mathit{notChosen}(S, P) \leftarrow \mathit{offer}(S, P), \mathit{not}\ \ \mathit{chosen}(S, P);$
(6) $\mathit{supplied}(S, P) \leftarrow \mathrm{DL}[\mathit{supplier} \uplus \mathit{chosen};\ \mathit{supplier}](S, P), \mathit{needed}(P);$
(7) $\mathit{anySupplied}(P) \leftarrow \mathit{supplied}(S, P), \mathit{needed}(P);$
(8) $\mathit{fail} \leftarrow \mathit{not}\ \ \mathit{fail}, \mathit{needed}(P), \mathit{not}\ \ \mathit{anySupplied}(P).$

Rule (2) extracts information on which parts already have a fixed vendor assigned from the DL, whereas Rule (3) imports the available offers for the needed parts not yet assigned. Rules (4)-(5) nondeterministically decide whether an offer should be chosen. Rule (6) summarizes the purchasing results by first sending the chosen assignments of vendors and parts from the LP-part to the ontology, and then querying for the overall $\mathit{supplier}$ relation. Finally, Rules (7)-(8) ensure that for every needed part there is a vendor chosen who supplies it. Note that Rule (8) acts as a constraint where the occurrences of the auxiliary atom fail in both, head and positive body, prevents all interpretations containing $\mathit{needed}(t)$ but not $\mathit{anySupplied}(t)$ for any term t from being an answer set. As we will see in Section 7.2.2, Φ_{ex} has one intended and one counterintuitive answer set.

We will now look at the semantics of our example DL-program in order to illustrate the core problem that is tackled in our approach.

Example 8. *$\mathcal{KB}_{ex}$ has two answer sets, I_1 and I_2, both containing the same atoms of predicates $needed$, $offer$, $alreadyContracted$, and $anySupplied$:*

$$\begin{aligned} I' = \{ \; & needed(cpu), needed(harddisk), needed(case), alreadyContracted(case), \\ & offer(s_1, cpu), offer(s_2, cpu), offer(s_3, harddisk), \\ & anySupplied(cpu), anySupplied(harddisk), anySupplied(case)\} \end{aligned}$$

The remaining atoms of I_1 are given by

$$\begin{aligned} I_1 \setminus I' = \{ & chosen(s_1, cpu), chosen(s_3, harddisk), notChosen(s_2, cpu), \\ & supplied(s_1, cpu), supplied(s_3, harddisk), supplied(s_3, case)\} \;, \end{aligned}$$

expressing a solution where the cpu is provided by shop s_1, whereas harddisk and case are delivered by vendor s_3.

The second answer set might seem surprising at first sight:

$$\begin{aligned} I_2 \setminus I' = \{ & chosen(s_2, cpu), chosen(s_3, harddisk), notChosen(s_2, cpu), \\ & supplied(s_1, cpu), supplied(s_1, harddisk), supplied(s_1, case), \\ & supplied(s_2, cpu), supplied(s_2, harddisk), supplied(s_2, case), \\ & supplied(s_3, cpu), supplied(s_3, harddisk), supplied(s_3, case), \\ & supplied(cpu, cpu), supplied(cpu, harddisk), supplied(cpu, case), \\ & supplied(harddisk, cpu), supplied(harddisk, harddisk), \\ & supplied(harddisk, case), supplied(case, cpu), supplied(case, harddisk), \\ & supplied(case, case)\} \end{aligned}$$

Apparently a situation is described in which each of the shops s_1, s_2, and s_3 supplies each of the needed hardware parts cpu, $case$, and $harddisk$, although the intention was that only a single shop supplies one part. Moreover, we also have atoms like $supplied(cpu, harddisk)$ in I_2, completely lacking intuition, as the first argument of predicate $supplied$ is supposed to refer to vendors only. The reason for the unintuitive results lies in an inconsistency emerging in the combination of the ontology and the logic programming part of $\mathcal{KB}_{ex}$. Note that atom $chosen(s_2, cpu) \in I_2$ suggests that shop s_2 has been chosen to deliver the cpu, although this shop is identified as disapproved in the DL-part (cf. Example 7). Consider any ground instance a' of DL-atom $a = \mathrm{DL}[supplier \uplus chosen; supplier](S, P)$ in Rule (6) of extended logic program Π_{ex}. We then have $\tau^I(a') = \{supplier(\mathbf{e}) \mid chosen(\mathbf{e}) \in I\}$ and therefore $supplier(s_2, cpu) \in \tau^I(a')$. As a consequence, $\Phi \cup \tau^I(a')$ is inconsistent since $\neg supplier(s_2, cpu)$ follows from the axioms $\exists supplier.\top \sqcap disapproved \sqsubseteq \bot$ and $disapproved(s_2)$ in Φ_{ex}. Due to this inconsistency every ground instance of a is true under I_2.

Whenever information, passed from the logic programming part Π to the ontology Φ of a DL-program, is inconsistent with Φ , unintuitive answer sets may arise as a consequence of trivial satisfaction of DL-atoms. In such cases we call the respective DL-atom *DL-inconsistent*.

Definition 20. *Let $\mathcal{KB} = (\Phi, \Pi)$ be a DL-program and I an interpretation relative to* Π. *A ground DL-atom $a = \mathrm{DL}[\gamma;\ Q](\mathbf{c})$ is* DL-consistent *under I w.r.t. Φ , if (1) $\Phi \models Q(\mathbf{c})$ or (2) $\Phi \cup \tau^I(a)$ is consistent, otherwise a is* DL-inconsistent *under I w.r.t. Φ .*

Intuitively, we are interested in avoiding using rules that have DL-inconsistent atoms in their bodies. Note that we use a notion of "inconsistency" that pertains to updates of the ontology: if some atom $Q(\mathbf{c})$ is entailed by the original ontology, we assume it is DL-consistent, even if updates via γ make the ontology inconsistent. Indeed, if $\Phi \models Q(\mathbf{c})$, we also have $\Phi \cup \tau^I(a) \models Q(\mathbf{c})$ for any update $\tau^I(a)$ due to monotonicity of usual Description Logics. If we would not take this case into account, we would disregard the whole rule (as seen in Definition 25).

7.3 Diagnosis of DL-calls

In what follows we are interested in finding diagnoses for inconsistency in the sense of Reiter [54], i.e., finding minimal sets of components of a faulty system such that the system would behave correctly if the components were. In our setting we will regard ground DL-atoms, that we refer to as *DL-calls*, as basic components since they constitute the interface between the logic programming and the Description Logics parts. Hence, given a DL-program $\mathcal{KB} = (\Phi, \Pi)$ that has no answer set, we are interested in minimal sets of DL-calls such that if the boolean result of the query of the DL-call is inverted, $\mathcal{KB}$ has an answer set.

Definition 21. *A* DL-call *of a DL-program $\mathcal{KB}$ is a DL-atom $l \in B(r)^+ \cup B(r)^-$ occurring in some DL-rule $r \in gr(\Pi)$. By $CALL(\mathcal{KB})$ we denote the set of all DL-calls of $\mathcal{KB}$.*

Note that, since DL-atoms have at most arity 2, they have only polynomial many ground instantiations.

Lemma 1. *Let $\mathcal{KB}$ be a DL-program. Then the number of DL-calls of $\mathcal{KB}$ is polynomial in the size of $\mathcal{KB}$.*

Definition 22. *Given a ground DL-atom $a = \mathrm{DL}[\gamma;\ Q](\mathbf{c})$, by $\neg a$ we denote the* inverse DL-atom $\mathrm{DL}[\gamma;\ \neg Q](\mathbf{c})$ *of a.*

Next, we formalize what we referred to as inverting of the boolean result of DL-calls by means of a program transformation.

Definition 23. *Let $\mathcal{KB} = (\Phi, \Pi)$ be a DL-program and $\mathcal{D}$ a set of DL-calls of $\mathcal{KB}$. By $\mathcal{KB}^{\neg}_{\mathcal{D}}$ we denote the* inverse grounding (Φ, Π') *of $\mathcal{KB}$ relative to $\mathcal{D}$, where Π' is obtained from $gr(\Pi)$ by replacing every occurrence of some $a \in \mathcal{D}$ by $\neg a$.*

Based on the introduced notions we can define a diagnosis of an inconsistent DL-program as follows.

Definition 24. *Let $\mathcal{KB}$ be an inconsistent DL-program. A set $\mathcal{D}$ of DL-calls of $\mathcal{KB}$ such that $\mathcal{KB}_{\mathcal{D}}^{\neg}$ is consistent is called a* diagnosis candidate *of $\mathcal{KB}$. Moreover, $\mathcal{D}$ is a* diagnosis *of $\mathcal{KB}$ if it is a subset minimal diagnosis candidate of $\mathcal{KB}$.*

7.3.1 Computing Diagnosis and Complexity

Next, we are concerned about computational aspects of diagnosis. That is, we discuss how to find diagnoses and the corresponding computational complexity.

A straightforward way of computing diagnosis candidates is by means of program transformations, using the formalism of DL-programs itself. Given a $\mathcal{KB} = (\Phi, \Pi)$ we introduce for every DL-atom $a(\mathbf{t}) \in \mathrm{D}(\mathcal{KB})$ of $\mathcal{KB}$ three fresh atoms $a'(\mathbf{t})$, $ab_a(\mathbf{t})$, and $no_a(\mathbf{t})$. We then define the translations $\mathcal{T}_1(\cdot)$ and $\mathcal{T}_2(\cdot)$ from a DL-program to another DL-program as $\mathcal{T}_1(\mathcal{KB}) = (\Phi, \mathcal{T}_1(\Pi))$, respectively $\mathcal{T}_2(\mathcal{KB}) = (\Phi, \mathcal{T}_2(\Pi))$, where $\mathcal{T}_1(\Pi)$ is obtained from Π by replacing every occurrence of some $a(\mathbf{t}) \in \mathrm{D}(\mathcal{KB})$ by $a'(\mathbf{t})$, $\mathcal{T}_2(\Pi) = \mathcal{T}_1(\Pi) \cup P$, and

$$\begin{aligned} P = \{ \quad & ab_a(\mathbf{t}) \leftarrow not\ no_a(\mathbf{t}), \quad no_a(\mathbf{t}) \leftarrow not\ ab_a(\mathbf{t}), \\ & a'(\mathbf{t}) \leftarrow a, not\ ab_a(\mathbf{t}), \quad a'(\mathbf{t}) \leftarrow not\ a, ab_a(\mathbf{t}) \mid a \in \mathrm{D}(\mathcal{KB})\}. \end{aligned}$$

We get the following correspondences.

Proposition 1. *Let $\mathcal{KB}$ be an inconsistent DL-program. $\mathcal{KB}$ has a diagnosis iff $\mathcal{T}_2(\mathcal{KB})$ is consistent.*

Proposition 2. *Let $\mathcal{KB}$ be an inconsistent DL-program and $S = \{\{a \mid ab_a(\in)I\} \mid I \in \mathrm{AS}(\mathcal{T}_2(\mathcal{KB}))\}$. Then the subset minimal sets in S correspond to the diagnoses of $\mathcal{KB}$.*

Using the translations defined above we can establish the first complexity result.

Theorem 1. *Let $\mathcal{KB} = (\Phi, \Pi)$ be an inconsistent DL-program where query answering in Φ is in complexity class* C*, deciding whether $\mathcal{KB}$ has a diagnosis is in* $\mathrm{NP}^{\mathrm{NExpTime} \cup \mathrm{C}}$.

Proof. First we guess a set $\mathcal{D}$ of DL-calls of $\mathcal{KB}$. Then, we evaluate every of the polynomially many DL-calls of $\mathcal{KB}$ using a C-oracle, and keep the DL-calls evaluated to true in the set $CALL^+$. Then the DL-program $(\emptyset, \mathcal{T}_1(\Pi) \cup \{a'(\mathbf{c}) \mid a(\mathbf{c}) \in (CALL^+ \setminus \mathcal{D}) \cup (\mathcal{D} \setminus CALL^+)\})$ that is free of DL-atoms is computed and we check whether it is consistent using a NExpTime-oracle. If so, some diagnosis $\mathcal{D}' \subseteq \mathcal{D}$ exists. □

We obtain a completeness result when query answering is known to be in possible in exponential time.

Lemma 2. *Let $\mathcal{KB} = (\Phi, \Pi)$ be a DL-program. Then, there exists a diagnosis for the DL-program $(\Phi, \mathcal{T}_2(\Pi) \cup \{\leftarrow ab_a(X)\} \mid a \in \mathrm{D}(\mathcal{KB}) \cup \{\leftarrow \mathrm{DL}[; \bot \sqsubseteq \top]\})$ iff $\mathcal{KB}$ is consistent.*

Theorem 2. *Let $\mathcal{KB} = (\Phi, \Pi)$ be an inconsistent DL-program where query answering in Φ is possible in exponential time, then deciding whether $\mathcal{KB}$ has a diagnosis is* NEXPTIME*-complete.*

Proof. Hardness follows from Lemma 2 and NEXPTIME-hardness of answer-set existence [20]. □

For positive DL-programs we get the following complexity result.

Theorem 3. *Let $\mathcal{KB} = (\Phi, \Pi)$ be an inconsistent positive DL-program where query answering in Φ is in complexity class* C*, deciding whether $\mathcal{KB}$ has a diagnosis is in* $\text{EXPTIME} \cup \text{P}^{\text{C}}$.

Proof. First we evaluate every of the polynomially many DL-calls of $\mathcal{KB}$ using a C-oracle, and keep the DL-calls evaluated to true in the set $CALL^{+}$. For every of the exponentially many $\mathcal{D} \subseteq CALL(\mathcal{KB})$, we compute the positive DL-program $(\emptyset, \mathcal{T}_1(\Pi) \cup \{a'(\mathbf{c}) \mid a(\mathbf{c}) \in (CALL^{+} \setminus \mathcal{D}) \cup (\mathcal{D} \setminus CALL^{+})\})$ that is free of DL-atoms and check whether it is consistent in exponential time. If so, some diagnosis $\mathcal{D}' \subseteq \mathcal{D}$ exists. □

Next, we investigate the complexity of checking whether a particular DL-call is contained in a diagnosis.

Theorem 4. *Let $\mathcal{KB} = (\Phi, \Pi)$ be an inconsistent DL-program where query answering in Φ is in complexity class* C*, and consider a DL-call a for $\mathcal{KB}$. Deciding whether a is contained in a diagnosis of $\mathcal{KB}$ is in* $\text{NP}^{\text{NEXPTIME} \cup \text{C}}$.

Proof. We proceed as in the proof of Theorem 1. We just need a further NEXPTIME-oracle call for checking consistency of the DL-program

$$\begin{aligned}\mathcal{KB}' = (\emptyset,\ & \mathcal{T}_1(\Pi) \cup \\ & \{ab_a(\mathbf{c}) \leftarrow not\ no_a(\mathbf{c}),\quad no_a(\mathbf{c}) \leftarrow not\ ab_a(\mathbf{c}) \mid a(\mathbf{c}) \in CALL(\mathcal{KB})\} \cup \\ & \{\leftarrow ab_a(\mathbf{c}) \mid a(\mathbf{c}) \in CALL(\mathcal{KB}) \setminus \mathcal{D}\} \cup \\ & \{\leftarrow \{ab_a(\mathbf{c}) \mid a(\mathbf{c}) \in \mathcal{D}\}\} \cup \\ & \{a'(\mathbf{c}) \leftarrow not\ ab_a(\mathbf{c}) \mid a(\mathbf{c}) \in CALL^{+}\} \cup \\ & \{a'(\mathbf{c}) \leftarrow ab_a(\mathbf{c}) \mid a(\mathbf{c}) \in CALL(\mathcal{KB}) \setminus CALL^{+}\}).\end{aligned}$$

The idea is that $\mathcal{KB}'$ guesses a combination of abnormal atoms for a set of DL-calls that is strictly smaller than $\mathcal{D}$. If 'reversing' the boolean values for these DL-calls establishes an answer set of $\mathcal{KB}$, we have an answer set for $\mathcal{KB}'$. If the oracle returns 'yes' the branch fails as then $\mathcal{D}$ is no diagnosis for not being minimal. Otherwise, the computation succeeds. □

Finally, we state a corresponding result for positive DL-programs.

Theorem 5. *Let $\mathcal{KB} = (\Phi, \Pi)$ be an inconsistent positive DL-program where query answering in Φ is in complexity class* C*, and consider a DL-call a for $\mathcal{KB}$. Deciding whether*

a is contained in a diagnosis of $\mathcal{KB}$ is in $\mathrm{P}^{\mathrm{C}} \cup$ EXPTIME. *In case query answering in Φ is* EXPTIME*-complete also deciding whether a is contained in a diagnosis of $\mathcal{KB}$ is* EXPTIME*-complete.*

Proof. For membership in case of an arbitrary C, we proceed as in the proof of Theorem 3, for computing all sets $\mathcal{D} \subseteq CALL(\mathcal{KB})$ such that some diagnosis $\mathcal{D}' \subseteq \mathcal{D}$ exists. For every such $\mathcal{D}$ we additionally check whether for all $\mathcal{D}' \subset \mathcal{D}$ the positive DL-program $(\emptyset, \mathcal{T}_1(\Pi) \cup \{a'(\mathbf{c}) \mid a(\mathbf{c}) \in (CALL^+ \setminus \mathcal{D}') \cup (\mathcal{D}' \setminus CALL^+)\})$ is inconsistent in exponential time. If so, no $\mathcal{D}' \subset \mathcal{D}$ is a diagnosis and hence $\mathcal{D}$ is one. We show hardness when query answering in Φ is EXPTIME-complete, by a reduction from query answering in Φ. Given a query $Q(\mathbf{c})$, some DL-call $\mathrm{DL}[;\, Q](\mathbf{c})$ is in a diagnosis of the inconsistent positive DL-program $(\Phi, \{h \leftarrow \mathrm{DL}[;\, \bot \sqsubseteq \top], \quad h \leftarrow \mathrm{DL}[;\, Q](\mathbf{c}), \quad \neg h\})$, where h is a fresh propositional atom, iff $Q(\mathbf{c})$ holds. □

7.4 Actions: DL-Inconsistency Tolerant Semantics

In what follows we introduce and discuss a refined semantics for DL-programs that limits the negative side effects of DL-inconsistency. It was developed within the ONTORULE project and published at the Extended Semantic Web Conference 2010 [51]. After introducing the new semantics, we provide some of its central properties and discuss strategies for implementation. Moreover, we define a stratification property that guarantees the uniqueness of answer sets under the new semantics and EXPTIME-completeness of deciding answer set existence. Based on these results, we present an algorithm for computing the answer set of a stratified program whenever one exists. We also analyze the complexity of deciding whether a DL-program has an answer set under the new semantics.

The central idea of the approach is to deactivate a rule whenever a DL-atom contained in its body becomes DL-inconsistent, in order to behave *tolerant* in the sense that flawed information does not influence the derived results. This way literals with unexpected argument types such as $supplied(cpu, harddisk)$ in Example 8, can be avoided in the information flow from the ontology to the logic program.

Definition 25. *Let $\mathcal{KB} = (\Phi, \Pi)$ be a DL-program and I an interpretation. I* t-satisfies *the body of a ground DL-rule r under Φ , denoted $I \not\models^{\Phi} B(r)$ if $I \models^{\Phi} B(r)$ and all DL-atoms in $B(r)$ are DL-consistent under I w.r.t. Φ . Moreover, I* t-satisfies *r under Φ , symbolically $I \not\models^{\Phi} r$, if $I \not\models^{\Phi} B(r)$ implies that $I \models^{\Phi} H(r)$. I is a* t-model *of a set Q of ground DL-rules under Φ denoted $I \not\models^{\Phi} Q$ if $I \not\models^{\Phi} r$ for all $r \in Q$. Finally, I is a* t-model *of $\mathcal{KB}$, denoted $I \not\models \mathcal{KB}$, if $I \not\models^{\Phi} gr(\Pi)$.*

Note that every model of $\mathcal{KB}$ is also a t-model of $\mathcal{KB}$. Moreover, if DL-atoms occur only in the negative bodies of rules in Π, also the converse holds. The reason for the latter is that a rule that is not applicable under DL-inconsistency tolerant semantics only because of a

DL-inconsistent DL-atom $a \in B(r)^-$ for some rule $r \in \Pi$ would also not be applicable under standard semantics as a would be satisfied as a consequence of DL-inconsistency.

Example 9. *Consider the ground instantiation*

$$r = \mathit{supplied}(\mathit{cpu}, \mathit{harddisk}) \leftarrow \mathrm{DL}[\mathit{supplier} \uplus \mathit{chosen};\ \mathit{supplier}](\mathit{cpu}, \mathit{harddisk}), \\ \mathit{needed}(\mathit{harddisk})$$

of Rule (6) of our running example. For interpretation I_2, as defined in Example 8, we have that $I_2 \models^{\Phi} B(r)$ but, as the DL-atom in $B(r)$ is DL-inconsistent under I_2 w.r.t. Φ_{ex}, it holds that $I_2 \not\models^{\Phi} B(r)$. As $H(r) \in I_2$, both $I_2 \models^{\Phi} r$ and $I_2 \not\models^{\Phi} r$. More general, since I_2 is a model of $\mathcal{KB}_{ex}$ it is also a t-model of $\mathcal{KB}_{ex}$. However, as we will see next, I_2 is not a t-answer set *of $\mathcal{KB}_{ex}$.*

For defining the notion of a t-answer set, we first give a modified version of the FLP-reduct, called *t-reduct*.

Definition 26. *Let $\Sigma = \langle \mathbf{C}, \mathbf{R}, \mathbf{P}, \mathbf{I} \rangle$ be a signature for DL-programs, Φ a DL knowledge base over $\langle \mathbf{I}, \mathcal{P}_o \rangle$, Π a set of ground DL-rules over $\Sigma_p = \langle \mathbf{P}, \mathbf{I} \rangle$, and I an interpretation over Σ_p. The* t-reduct $\Pi_t^{I,\Phi}$ *of Π under Φ relative to I is the set of rules $r \in \Pi$ such that $I \not\models^{\Phi} B(r)$. Moreover, the* t-reduct $\mathcal{KB}_t^I$ *of a (possibly non-ground) DL-program $\mathcal{KB} = (\Phi, \Pi)$ relative to I is given by $gr(\Pi)_t^{I,\Phi}$.*

Definition 27. *Let $\mathcal{KB}$ be a DL-program. An interpretation I is a* t-answer set *of $\mathcal{KB}$, if I is a subset-minimal t-model of $\mathcal{KB}_t^I$. The set of all t-answer sets of $\mathcal{KB}$ is denoted by* $\mathrm{AS}^{\mathrm{t}}(\mathcal{KB})$.

Example 10. *For the program $\mathcal{KB}_{ex}$ of the product database example, the only t-answer set is given by interpretation I_1, as defined in Example 8. As stated in Example 9, I_2 is a t-model of $\mathcal{KB}_{ex}$; however, I_2 is not a minimal t-model of $(\mathcal{KB}_{ex})_t^{I_2}$, as required in Definition 27 for being a t-answer set. In fact, the ground instance of Rule (6) in Example 9 is not contained in $(\mathcal{KB}_{ex})_t^{I_2}$. Therefore, we can remove the head of the rule, atom $\mathit{supplied}(\mathit{cpu}, \mathit{harddisk})$, from I_2 such that the resulting interpretation I_2' is still a t-model of $\mathcal{KB}_{ex}$.*

Whenever no DL-atoms are present in a DL-program $\mathcal{KB} = (\Phi, \Pi)$, DL-inconsistency tolerant semantics reduces to answer set semantics of the ordinary logic program Π. Therefore, the next result is a proper extension to a similar one that is folklore for standard logic programs.

Theorem 6. *For every t-answer set I of a DL-program $\mathcal{KB}$, I is a minimal t-model of $\mathcal{KB}$.*

Note that the converse does not generally hold. For example, consider the set

$$I_3 = I' \cup \{\mathit{chosen}(s_2, \mathit{cpu}), \mathit{chosen}(s_3, \mathit{harddisk}), \mathit{notChosen}(s_2, \mathit{cpu}), \\ \mathit{notChosen}(s_2, \mathit{harddisk})\}\ ,$$

where I' is given as in Example 8. I_3 is a minimal t-model of $\mathcal{KB}_{ex}$ but, since the ground instantiation

$$\mathit{fail} \leftarrow \mathit{not\ fail}, \mathit{needed}(\mathit{cpu}), \mathit{not\ anySupplied}(\mathit{cpu})$$

of Rule (8) from our example is not contained in $(\mathcal{KB}_{ex})_t^{I_3}$, we can remove atom $\mathit{anySupplied}(\mathit{cpu})$ from I_3 such that the resulting interpretation I_3' is still a t-model of $(\mathcal{KB}_{ex})_t^{I_3}$. Consequently, by Definition 27, I_3 is no t-answer set of $\mathcal{KB}_{ex}$.

The next result relates the refined semantics to the FLP-semantics.

Proposition 3. *Let $\mathcal{KB} = (\Phi, \Pi)$ be a monotonic DL-program and let I be an answer set of $\mathcal{KB}$. If all DL-atoms in $gr(\Pi)$ are DL-consistent under I w.r.t. Φ , then I is a t-answer set of $\mathcal{KB}$.*

Note that DL-programs with no occurrences of the $\Cap$ operator are monotonic and, as remarked in Section 7.1, this operator can typically be avoided in applications.

While counterintuitive literals a là $\mathit{supplied}(\mathit{cpu}, \mathit{harddisk})$ cannot occur in a t-answer set, Proposition 3 suggests that results that are intuitive are preserved under the refined semantics, as answer sets of a DL-program where inconsistency is immaterial are selected. On the other hand, a DL-program may have t-answer sets that do not correspond to any answer set (due to inconsistency avoidance).

Example 11. *Consider the DL-program $\mathcal{KB} = (\Phi, \Pi)$ where $\Phi = \{\neg C(a)\}$ and $\Pi = \{\, p(a);\ \ \mathit{fail} \leftarrow \mathit{not\ fail}, \mathrm{DL}[C \uplus p;\ C](a)\,\}$. Clearly, $\mathcal{KB}$ has no answer set, as the DL-atom in Π is DL-inconsistent; its single t-answer set is $I = \{p(a)\}$.*

7.4.1 Computational Aspects

Translation to FLP Semantics The DL-inconsistency tolerant semantics of DL-programs can be simulated by the FLP semantics as in Definition 17 using a linear rule-by-rule transformation $\rho(\cdot)$ on generalized normal programs, defined as

$$\begin{aligned}
\rho(\Pi) =& \{\rho(r) \mid r \in \Pi\} \cup \\
& \{a' \leftarrow \mathrm{DL}[\gamma;\ \top \sqsubseteq \bot], \mathit{not}\ \mathrm{DL}[;\ Q](\mathbf{t}) \mid r \in \Pi, \mathit{not}\ a \in A(r)\}, \quad \text{where} \\
\rho(r) =& H(r) \leftarrow B(r) \cup A(r), \quad \text{and} \\
A(r) =& \{\mathit{not}\ a' \mid a = \mathrm{DL}[\gamma;\ Q](\mathbf{t}) \in B(r)\}.
\end{aligned}$$

In the translation for each DL-atom $a = \mathrm{DL}[\gamma;\ Q](\mathbf{t})$ occurring in the body of a rule r, we add a new atom a' to the negative body of r and a rule that deduces a' exactly when $I \models^{\Phi} \mathrm{DL}[\gamma;\ \top \sqsubseteq \bot]$ and $I \not\models^{\Phi} \mathrm{DL}[;\ Q](\mathbf{t})$ for some interpretation I, i.e., when $\Phi \cup \tau^I(a)$ is inconsistent and $\Phi \not\models Q(\mathbf{c})$, and thus a is DL-consistent. Deduction of a' thus causes the body of the transformed rule to be false under FLP-semantics corresponding exactly the case where the atom a is DL-inconsistent. Thus for a rule r in Π under a DL-inconsistency tolerant semantics and its corresponding rule $\rho(r)$ in the transformed program $\rho(\Pi)$ under FLP-semantics, we have that r and $\rho(r)$ have bodies whose truth values correspond under the respective semantics, thus effectively mimicking the DL-inconsistency tolerant semantics with the FLP-semantics.

Theorem 7. *For every DL-program* $\mathcal{KB} = (\Phi, \Pi)$*,* $\mathrm{AS}^{\mathrm{t}}(\mathcal{KB}) = \{I \cap \mathrm{HB}(\Pi) \mid I \in \mathrm{AS}((\Phi, \rho(\Pi)))\}$.

By means of this translation, the t-answer sets of $\mathcal{KB}$ can be computed utilizing DLVHEX, a solver for *non-monotonic logic programs* admitting *higher-order atoms* and *external atoms*, or *HEX-programs* for short [20], that have a semantics based on the FLP-reduct. A plug-in for evaluating DL-programs, without the $\Cap$ operator, is available for DLVHEX that gives access to the DL-knowledge base by means of a third-party DL-reasoner [19, 38].

Due to the close relationship to HEX-programs, results on their computational complexity carry over to DL-inconsistency tolerant semantics of DL-programs. In particular, as corollaries of Theorem 7 and 8 in [20], due to the existence of transformation $\rho(\cdot)$, we obtain the following two results.

Theorem 8. *Given a DL-program* $\mathcal{KB} = (\Phi, \Pi)$*, where query answering in* Φ *is in complexity class* C*, deciding whether* $\mathcal{KB}$ *has a t-answer set is in* $\mathrm{NE{\scriptstyle XP}T{\scriptstyle IME}}^{\mathrm{C}}$.

Theorem 9. *Given a DL-program* $\mathcal{KB} = (\Phi, \Pi)$*, where query answering in* Φ *is in* EXPTIME*, deciding whether* $\mathcal{KB}$ *has a t-answer set is* NEXPTIME*-complete.*

Hardness in Theorem 9 follows from the special case of DL-programs without any DL-atoms, for which the DL-inconsistency tolerant semantics reduces to the standard answer set semantics of normal logic programs. It is known that answer set existence for this class of programs is NEXPTIME-complete. On the other side, membership follows again from the translation to HEX-programs, as it is known that checking the answer sets of HEX-programs is NEXPTIME-complete under the restriction that the external atoms can be evaluated in exponential time [20].

The result is especially interesting as query answering is in EXPTIME for many important Description Logics such as the basic DL $\mathcal{ALC}$, the DL underlying OWL-Lite ($\mathcal{SHIF}$), and the Description Logics corresponding to the fragments OWL 2 EL, OWL 2 RL, OWL 2 QL of the upcoming standard for a Web Ontology Language [44].

Another important aspect of the complexity results is that for DL-programs, reasoning under DL-inconsistency tolerant semantics is not harder than under the FLP-semantics.

Stratification Eiter et al. [18] defined an iterative least model semantics for DL-programs that have a certain stratification property (which we will here refer to as *standard stratification*). The idea of stratification is to layer a program into a number of ordered strata that can be efficiently evaluated one-by-one where lower strata do not depend on higher strata.

A DL-program $\mathcal{KB}$ which is standard stratified has at most one answer set that coincides with its iterative least model and conversely, if $\mathcal{KB}$ has an iterative least model it coincides with the unique answer set of $\mathcal{KB}$. However, a DL-program that is standard stratified may have multiple t-answer sets. Too see this, note that a positive DL-program always has a standard stratification with a single stratum. Consider, e.g., the DL-program $\mathcal{KB} = (\Phi, \Pi)$, with

$$\begin{aligned}\Pi = \{\ & h(c),\\ & a(c) \leftarrow \mathrm{DL}[B \uplus b, H \uplus h;\ H](c),\\ & b(c) \leftarrow \mathrm{DL}[A \uplus a, H \uplus h;\ H](c)\}\ ,\end{aligned}$$

where $\Phi \not\models A(c)$ and $\Phi \not\models B(c)$. This program has two t-answer sets, viz. $I_1 = \{a(c), h(c)\}$ and $I_2 = \{b(c), h(c)\}$.

In the following, we define a different kind of stratification (which we call *t-stratification*) that guarantees a unique t-answer set iff the respective t-stratified program has a t-answer set. The major difference to standard stratification is to enforce that the information necessary for evaluating DL-atoms must be already available on a strictly lower stratum then the current one during a computation.

Definition 28. *A* t-stratification *of a DL-program* $\mathcal{KB} = (\Phi, \Pi)$ *is a mapping* $\mu : \mathrm{HB}(\Pi) \cup \mathrm{D}(\Pi) \rightarrow \{0, 1, \ldots, k\}$, *where* $\mathrm{D}(\Pi)$ *is the set of DL-atoms occurring in* $gr(\Pi)$, *such that*

(i) for each $r \in gr(\Pi)$, $\mu(H(r)) \geq \mu(l')$ *for all* $l' \in B(r)^+$, $\mu(H(r)) > \mu(l')$ *for all* $l' \in B(r)^-$, *and* $\mu(H(r)) > \mu(l')$ *for each DL-atom* $l' \in B(r)$, *and*

(ii) $\mu(a) \geq \mu(l)$ *for each input literal* l *of each DL-atom* $a \in \mathrm{D}(\Pi)$.

We call $k \geq 0$ the length of μ. For every $i \in \{0, \ldots, k\}$, we then define the DL-programs $\mathcal{KB}_{\mu,i}$ as (Φ, Π_i), where $\Pi_i = \{r \in gr(\Pi) \mid \mu(H(r)) = i\}$ and $\mathcal{KB}^*_{\mu,i}$ as $(\Phi, \Pi^*_{\mu,i})$ where $\Pi^*_{\mu,i} = \{r \in gr(\Pi) \mid \mu(H(r)) \leq i\}$. Likewise, we define $\mathrm{HB}_{\mu,i}(\Pi)$ (resp., $\mathrm{HB}^*_{\mu,i}(\Pi)$) as the set of all $l \in \mathrm{HB}(\Pi)$ such that $\mu(l) = i$ (resp., $\mu(l) \leq i$). We say that a DL-program $\mathcal{KB}$ is *t-stratified*, if it has a t-stratification μ of length $k \geq 0$. It is easy to see that for DL-programs without DL-atoms, t-stratification reduces to standard stratification of logic programs. Moreover, checking whether a DL-program is t-stratified and computing a t-stratification can be done by modified algorithms for standard stratification in linear time.

Note that by Definition 28, $\mathcal{KB}^*_{\mu,0}$ is always a positive DL-program without DL-atoms. Consequently, Π_0 coincides with a positive logic program, for which DL-inconsistency tolerant semantics coincides with the answer set semantics of logic programs. Therefore, the following proposition holds.

Proposition 4. *Let* $\mathcal{KB}$ *be a DL-program* $\mathcal{KB} = (\Phi, \Pi)$ *with t-stratification* μ. *Then,* $\mathcal{KB}^*_{\mu,0}$ *has a unique minimal t-model that is also the unique t-answer set of* $\mathcal{KB}^*_{\mu,0}$.

Next we want to establish uniqueness of t-answer sets for arbitrary strata.

Lemma 3. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ. If I_1 and I_2 are t-answer sets of $\mathcal{KB}^*_{\mu,i}$ for $i \geq 0$, then $I_1 = I_2$.*

As a consequence of this lemma and Proposition 4, we get the next result.

Theorem 10. *Let $\mathcal{KB}$ be a t-stratified DL-program $\mathcal{KB} = (\Phi, \Pi)$. If $\mathcal{KB}$ has a t-answer set, then this t-answer set is unique.*

As can be seen in the next result, the t-answer set of a t-stratified DL-program is *compositional* in the sense that, roughly speaking, we get t-answer sets for the part of the DL-program that is below a certain stratum, if we remove all atoms of higher strata from I.

Theorem 11. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ. If I is a t-answer set of $\mathcal{KB}^*_{\mu,i}$ for $i > 0$, then $I \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)$ is a t-answer set of $\mathcal{KB}^*_{\mu,i-1}$.*

Approaching from this result, we aim at computing the t-answer set I of $\mathcal{KB}$ step-by-step, starting with $I \cap \mathrm{HB}^*_{\mu,0}(\Pi)$ and extending the interpretation one stratum a time until we reach $I = I \cap \mathrm{HB}^*_{\mu,k}(\Pi)$. Hence, we define a series of sets $\Delta_{i,h}$ for each stratum i, that can be seen as the results of repeatedly applying a consequence operator.

Definition 29. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ and I_{i-1} a t-answer set of $\mathcal{KB}^*_{\mu,i-1}$ for some $i > 0$. We define sets of literals $\Delta_{i,h}$ for $h \geq 0$ as follows:*

(i) $\Delta_{i,0} = \emptyset$ *and*

(ii) $\Delta_{i,m} = \bigcup_{o<m} \Delta_{i,o} \cup \{H(r) \mid \mu(H(r)) = i, I_{i-1} \cup \Delta_{i,m-1} \not\models^{\Phi} B(r)\}$ *for* $m > 0$.

As $gr(\Pi)$ contains only a finite number of rules, and $\Delta_{i,h} \subseteq \Delta_{i,h+1}$ for all h, we must always reach some fixpoint Δ_i. That is, $\Delta_i = \Delta_{i,f}$ when $\Delta_{i,f} = \Delta_{i,f+1}$.

In order to establish our main result on computing the unique t-answer set (whenever one exists), we make use of the following lemma.

Lemma 4. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ. If I_1 and I_2 are t-models of $\mathcal{KB}^*_{\mu,i}$ for $i \geq 0$ such that $I_1 \cap \mathrm{HB}^*_{\mu,i-1}(\Pi) = I_2 \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)$ then $I_1 \cap I_2$ is a t-model of $\mathcal{KB}^*_{\mu,i}$.*

Intuitively, when we can extend a t-model of lower strata of the DL-program to a further stratum, there is always a subset minimal extension of this t-model.

By computing the t-answer set of $\mathcal{KB}^*_{\mu,0}$ and subsequently Δ_i for each stratum i, we can compute the t-answer set of $\mathcal{KB}$, whenever it exists:

Theorem 12. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ and let I be a t-answer set of $\mathcal{KB}^*_{\mu,i}$ for some $i > 0$. Then, $I = I'$ where $I' = (I \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)) \cup \Delta_i$.*

Proof. Towards a contradiction assume $I \neq I'$. From Theorem 10 follows that $I' \notin \mathrm{AS}^{\mathrm{t}}(\mathcal{KB}^*_{\mu,i})$. As I is a minimal t-model of $\mathcal{KB}^*_{\mu,i}$, we get $I \cap I' \not\models^{\Phi} \Pi^*_{\mu,i}$. From this and Lemma 4 follows by modus tollens that $I' \not\models^{\Phi} \Pi^*_{\mu,i}$. Hence, there is a rule $r \in \Pi^*_{\mu,i}$ with $I' \not\models^{\Phi} B(r)$ and $I' \not\models^{\Phi} H(r)$. Consider the case that $\mu(H(r)) < i$. Then, $I' \not\models^{\Phi} r$ is a contradiction to $I \not\models^{\Phi} r$, since $I \cap \mathrm{HB}^*_{\mu,i-1}(\Pi) = I' \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)$. Now consider case $\mu(H(r)) = i$ and number $m \leq 0$ such that $\Delta_{i,m} = \Delta_i$. As $\Delta_{i,m} \subseteq I'$ and $I' \not\models^{\Phi} H(r)$, we have $H(r) \notin \Delta_{i,m}$. Moreover, since $I' \not\models^{\Phi} B(r)$ and $I' = (I \cap \mathrm{HB}^*_{\mu,i-1}(\Pi)) \cup \Delta_{i,m}$, by Definition 29 we have that $H(r) \in \Delta_{i,m+1}$. As then $\Delta_{i,m} \neq \Delta_{i,m+1}$, we have a contradiction to $\Delta_{i,m}$ being the fixpoint Δ_i. □

So far we established that in case there is a t-answer set we can compute it stratum by stratum. In the following, we provide means for deciding the existence of a t-answer set during this computation.

Theorem 13. *Let $\mathcal{KB}$ be a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ and I_{i-1} a t-answer set of $\mathcal{KB}^*_{\mu,i-1}$ for some $i > 0$. If $I_i = I_{i-1} \cup \Delta_i$ is a t-model of $\mathcal{KB}^*_{\mu,i}$ then I_i is a t-answer set of $\mathcal{KB}^*_{\mu,i}$.*

This enables us to pursue the following approach. After computing $I_i = I_{i-1} \cup \Delta_i$ for a stratum i, we check whether $I \not\models^{\Phi} \Pi^*_{\mu,i}$. If yes, we know by Theorem 13 that I_i is a t-answer set of $\mathcal{KB}^*_{\mu,i}$ and we are either done or continue our computation for stratum $i+1$. If $I \not\models^{\Phi} \Pi^*_{\mu,i}$, we know by Theorem 12 that $\mathcal{KB}^*_{\mu,i}$ has no t-answer set and stop the computation.

Algorithm 1 for computing the t-answer set of a given DL-program $\mathcal{KB}$ with a t-stratification follows precisely this strategy after having computed the unique t-answer set of $\mathcal{KB}^*_{\mu,0}$. This can be done by a standard answer set solver as $\mathcal{KB}^*_{\mu,0}$ does not involve DL-atoms. Overall, the algorithm runs in exponential time with an additional effort of external calls to a DL-reasoner for evaluating the DL-queries of DL-atoms in lines 10 and 16. The time necessary for this evaluations depends on the complexity of query answering in the respective DL. Altogether, there may be an exponential number of such calls.

Theorem 14. *Given a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ, where query answering in Φ is in complexity class* C*, deciding whether $\mathcal{KB}$ has a t-answer set is in* $\textsc{ExpTime}^{\mathrm{C}}$.

When query answering in Φ is possible in exponential time, in the worst case the algorithm has to perform an exponential number of exponential time calls which can in turn be done in exponential time.

Theorem 15. *Given a DL-program $\mathcal{KB} = (\Phi, \Pi)$ with t-stratification μ, where query answering in Φ is in* ExpTime*, deciding whether $\mathcal{KB}$ has a t-answer set is* ExpTime*-complete.*

Hardness follows from ExpTime-completeness of ordinary stratified logic programs. For lightweight Description Logics such as those underlying OWL 2 EL, OWL 2 RL, and

Algorithm 1 Computing the t-answer set of a t-stratified DL-program $\mathcal{KB}$

```
Require: $\mathcal{KB} = (\Phi, \Pi)$, $\mu$ is a t-stratification of $\mathcal{KB}$ of length $k \geq 0$
 1: $I_0 :=$ the unique t-answer set of $\mathcal{KB}^*_{\mu,0}$        // computable in exponential time
 2: for $i := 1$ to $k$ do
 3:   // compute $\Delta_i$
 4:   $\Delta' := \emptyset$
 5:   repeat
 6:     $\Delta_i := \Delta'$
 7:     for all $r \in gr(\Pi_i)$ do
 8:       // loop may have exponentially many iterations
 9:       // the following check requires two queries to $\Phi$ per DL-atom in $B(r)$:
10:       if $\Delta_i \cup I_{i-1} \not\models^{\Phi} B(r)$ then
11:         $\Delta' := \Delta' \cup \{H(r)\}$
12:       end if
13:     end for
14:   until $\Delta_i = \Delta'$      // number of iterations limited by number of rules in $gr(\Pi_i)$
15:   $I_i := I_{i-1} \cup \Delta_i$
16:   if $I_i \not\models^{\Phi} gr(\Pi^*_{\mu,i})$ then
17:     print "$\mathcal{KB}$ has no t-answer set."
18:     return
19:   end if
20: end for
21: return $I_k$               // $I_k$ is the unique t-answer set of $\mathcal{KB}$
```

OWL 2 QL, where query answering has polynomial data complexity, reasoning for DL-programs is feasible in polynomial time under data complexity (where all of $\mathcal{KB}$ except facts in Φ and Π is fixed).

Chapter 8

Conclusion

In this document we have reported the preliminary achievements towards consistency maintainance within the ONTORULE project. It will be succeeded and finalized by Deliverable 2.6 “Consistency maintenance. Final Report”.

The current achievements include a general consistency framework as a basis for communicating consistency related requirements and methods for the individual formalisms considered within the project. The framework defines common terminology, classifies consistency related problems, and identifies main tasks for consistency maintenance. On the problem side, four abstract problem types for classification of consistency problems have been presented. These are contradiction, completeness, relevance, and language conformance. We identified problems of these categories for rule and ontology languages used in ONTORULE and additionally, we examined inconsistencies in the context of texts. Here, we have an additional potential for inconsistencies caused by the ambiguities of natural language semantics. However, most problems that arise in formal rule languages can also occur in texts. One example are completeness problems such as case distinctions where not every case that may arise in practice is covered. As texts are the major source for acquisition in ONTORULE, inconsistency prevention should be done already in the natural language representation.

We have developed methods for diagnosing inconsistencies and anomalies for logical rules as well as production rules. In the case of logical rules we describe a declarative approach towards detecting anomalies in the ObjectLogic language which originates from F-Logic which is a key rule language in WP3. The method identifies typical cases of inconsistency using ObjectLogic itself. For production rules, we have introduced Change Management Patterns that classify different types of changes, inconsistencies. While some types of anomalies such as contradicting rules, subsumed rules, or rules whose antecedent cannot become true are shared by logical and production rules, the rich ObjectLogic language gives rise to further sources of inconsistency. In particular, ObjectLogic can also be seen as an ontology language and therefore also shares inconsistency problems with Description Logics such as concepts without instances. For both types of rule languages we identify actions to resolve the individual types of anomalies.

For ontology languages, we have examined the state-of-the-art on research into inconsistency in Description Logics. Most approaches focus on problems related to logical contradictions, i.e., inconsistency in the sense of logical unsatisfiability. Actions for handling inconsistencies include consistency restoring methods on the one hand, and inconsistency tolerant approaches on the other hand.

Besides methods for established formalisms we have also provided initial results on consistency maintenance in formalisms that combine ontologies and rules as developed in WP3. We have focussed on loosely coupling of Description Logics and logical rules realized by DL-programs. Results carry over to the related language of F-Logic# which was developed within WP3. We have tackled two consistency related phenomena that are intrinsic to the coupling mechanism. On the one hand, we have provided definitions and complexity analysis for diagnosing minimal sets of calls to the Description Logic knowledge base that can restore consistency of inconsistent DL-programs when reversed. On the other hand, we have introduced a semantics for DL-programs that tolerates certain inconsistencies due to the ontology update-feature of the loose-coupling mechanism.

Bibliography

[1] Vincenzo Ambriola and Vincenzo Gervasi. On the systematic analysis of natural language requirements with CIRCE. *Automated Software Engineering*, 13:107–167, 2006.

[2] Jürgen Angele, Michael Kifer, and Georg Lausen. Ontologies in F-Logic. In Peter Bernus, Jacek Błażewicz, Günter J. Schmidt, Michael J. Shaw, Steffen Staab, and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 45–70. Springer Berlin Heidelberg, 2009.

[3] F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning*, 14(2):149–180, 1995.

[4] Franz Baader, Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. Query and predicate emptiness in Description Logics. In Fangzhen Lin and Ulrike Sattler, editors, *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR'10), Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press, 2010.

[5] Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 20(1):5–34, 2010.

[6] Joachim Baumeister, Thomas Kleemann, and Dietmar Seipel. Towards the verification of ontologies with rules. In David Wilson and Geoff Sutcliffe, editors, *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference (FLAIRS'07), Key West, Florida, USA, May 7-9, 2007*, pages 524–529. AAAI Press, 2007.

[7] Joachim Baumeister and Dietmar Seipel. Anomalies in ontologies with rules. *Journal of Web Semantics*, 8(1):55–68, 2010.

[8] Nuel D. Belnap. A useful four-valued logic. In J. Michael Dunn and George Epstein, editors, *Modern Uses of Multiple-Valued Logics*, pages 8–37. Reidel, Dordrecht, 1977.

[9] Alex Borgida, Enrico Franconi, and Ian Horrocks. Explaining ALC subsumption. In Patrick Lambrix, Alexander Borgida, Maurizio Lenzerini, Ralf Möller, and Peter F.

Patel-Schneider, editors, *Proceedings of the 1999 International Workshop on Description Logics (DL'99), Linköping, Sweden, July 30 - August 1, 1999*, volume 22 of *CEUR Workshop Proceedings*, pages 33–36. CEUR-WS.org, 1999.

[10] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: a structural data model for objects. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD international conference on Management of data, Portland, Oregon, USA, May 31 - June 2, 1989*, SIGMOD '89, pages 58–67, New York, NY, USA, 1989. ACM.

[11] Carolyn Brodie, Clare-Marie Karat, and John Karat. An empirical study of natural language parsing of privacy policy rules using the SPARCLE policy workbench. In Lorrie Faith Cranor, editor, *Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS'06), Pittsburgh, Pennsylvania, USA, July 12-14, 2006*, volume 149 of *ACM International Conference Proceeding Series*, pages 8–19. ACM, 2006.

[12] Jos de Bruijn, Philippe Bonnard, Hugues Citeau, Sylvain Dehors, Stijn Heymans, Roman Korf, Jörg Pührer, and Thomas Eiter. D3.1 State-of-the-art survey of issues. Technical report, ONTORULE IST-2009-231875 Project, 2009.

[13] Xi Deng. *Explanation and Diagnosis Services for Unsatisfiability and Inconsistency in Description Logics*. PhD thesis, Concordia University, 2010.

[14] Xi Deng, Volker Haarslev, and Nematollaah Shiri. A resolution based framework to explain reasoning in Description Logics. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL'05), Edinburgh, Scotland, UK, July 26-28, 2005*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[15] Nikhil Dinesh, Aravind Joshi, Insup Lee, and Bonnie Webber. Extracting formal specifications from natural language regulatory documents. In *Proceedings of the Fifth International Workshop on Inference in Computational Semantics (ICoS-5), Buxton, England, UK, 2021 April, 2006*, pages 1–10, 2006.

[16] Rim Djedidi and Marie-Aude Aufaure. Change management patterns (CMP) for ontology evolution process. *Lecture Notes in Computer Science*, 5956(2-3):286–305, 2010.

[17] Norbert Eisinger. *Completeness, Confluence, and Related Properties of Clause Graph Resolution*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991.

[18] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with Description Logics for the Semantic Web. *Artificial Intelligence*, 172(12-13):1495–1539, 2008.

[19] Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner, and Roman Schindlauer. Exploiting conjunctive queries in description logic programs. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):115–152, 2008.

[20] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 90–96. Professional Book Center, 2005.

[21] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Well-founded semantics for description logic programs in the semantic web. In *Proceedings of the Third International Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML'04), Hiroshima, Japan, November 8, 2004*, volume 3323 of *Lecture Notes in Computer Science*, pages 81–97. Springer, 2004.

[22] Adil El Ghali, Sylvain Dehors, Roman Korf, Lévy François, and Abdoulaye Guissé. D2.1 Basic integration of ontology and business rule authoring technology and basic dependency management. Technical report, ONTORULE IST-2009-231875 Project, 2009.

[23] F. Fabbrini, M. Fusani, V. Gervasi, S. Gnesi, and S. Ruggieri. Achieving quality in natural language requirements. In *Eleventh International Software Quality Week*, S. Francisco, California, 1998. Software Research Institute.

[24] Wolfgang Faber, Nicola Leone, and Gerard Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In José Júlio Alferes and João Alexandre Leite, editors, *Proceedings of the European Conference on Logics in Artificial Intelligence (JELIA'04), Lisbon, Portugal, September 27-30, 2004*, volume 3229 of *Lecture Notes in Computer Science*. Springer, 2004. Revised version: http://www.wfaber.com/research/papers/jelia2004.pdf.

[25] Gerhard Friedrich and Kostyantyn M. Shchekotykhin. A general diagnosis method for ontologies. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *Proceedings of the Fourth International Semantic Web Conference,(ISWC'05), Galway, Ireland, November 6-10, 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2005.

[26] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[27] Gerhard Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39:176–210, 1935. 10.1007/BF01201353.

[28] Guido Governatori. Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14(2-3):181–216, 2005.

[29] Guido Governatori and Zoran Milosevic. A formal analysis of a business contract language. *International Journal of Cooperative Information Systems*, 15(4):659–685, 2006.

[30] Stijn Heymans, Jos de Bruijn, Martín Rezk, Hassan Aït-Kaci, Hugues Citeau, Roman Korf, Jörg Pührer, Cristina Feier, and Thomas Eiter. D3.2 Initial combinations of rules and ontologies. Technical report, ONTORULE IST-2009-231875 Project, December 2009. `http://ontorule-project.eu/wiki/InitialCombinations`.

[31] Ian R. Horrocks. Using an expressive Description Logic: FaCT or fiction? In Anthony G. Cohn, Lenhard K. Schubert, and Stuart C. Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998*, pages 636–647. Morgan Kaufmann, 1998.

[32] Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with inconsistent ontologies. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05), Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 454–459. Professional Book Center, 2005.

[33] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing SHIQ-description logic to disjunctive datalog programs. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *Proceedings of the Ninth International Conference (KR'04), Whistler, British Columbia, Canada, June 2-5, 2004*, pages 152–162, 2004.

[34] Åshild Johnsen and Arne-Jørgen Berre. A bridge between legislator and technologist - Formalization in SBVR for improved quality and understanding of legal rules. In *Proceedings of the First Workshop on Aided Formalization of Business Rules and Policies (FoBuR'10), Lisbon, Portugal, October 11-15, 2010*, 2010.

[35] Aditya Kalyanpur. *Debugging and repair of OWL ontologies*. PhD thesis, University of Maryland at College Park, College Park, Maryland, USA, 2006. AAI3222483.

[36] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics – Special Issue on the Semantic Web Track of WWW 2005*, 3(4), 2005.

[37] Fábio Natanael Kepler, Christian Paz-Trillo, Joselyto Riani, Márcio Moretto Ribeiro, Karina Valdivia Delgado, Leliane Nunes de Barros, and Renata Wassermann. Classifying ontologies. In Frederico Luiz Gonçalves de Freitas, Heiner Stuckenschmidt, Helena Sofia Pinto, and Andreia Malucelli, editors, *Proceedings*

of the 2nd Workshop on Ontologies and their Applications (WONTO'06), Ribeirao Preto, Brazil, October 23-27, 2006, volume 199 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[38] Thomas Krennwallner. Integration of conjunctive queries over Description Logics into HEX-programs. Master's thesis, Vienna University of Technology, October 2007.

[39] Kevin Lee, Thomas Meyer, Jeff Z. Pan, and Richard Booth. Computing maximally satisfiable terminologies for the Description Logic *lc* with cyclic definitions. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *Proceedings of the 2006 International Workshop on Description Logics (DL'06), Windermere, Lake District, UK, May 30 - June 1, 2006*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[40] F. Lévy, A. Guissé, A. Nazarenko, N. Omrane, and S. Szulman. An environment for the joint management of written policies and business rules. In Eric Gregoire, editor, *Proceedings of the Twenty-Second International Conference on Tools with Artificial Intelligence (ICTAI'10), Arras, France, October 27-29, 2010*, volume 2, pages 142–149. IEEE, 2010.

[41] Yue Ma and Pascal Hitzler. Paraconsistent reasoning for OWL 2. In Axel Polleres and Terrance Swift, editors, *Proceedings of the Third International Conference on Web Reasoning and Rule Systems (RR'09), Chantilly, Virginia, USA, October 25-26, 2009*, volume 5837 of *Lecture Notes in Computer Science*, pages 197–211. Springer, 2009.

[42] Yue Ma, Pascal Hitzler, and Zuoquan Lin. Algorithms for paraconsistent reasoning with OWL. In Enrico Franconi, Michael Kifer, Wolfgang May, Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *Proceedings of the Fourth European Semantic Web Conference (ESWC'07), Innsbruck, Austria, June 3-7, 2007*, volume 4519 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2007.

[43] Deborah L. McGuinness and Alex Borgida. Explaining subsumption in Description Logics. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95), Montréal, Québec, Canada, August 20-25, 1995*, pages 816–821. Morgan Kaufmann, 1995.

[44] Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. OWL 2 web ontology language structural specification and functional-style syntax. W3C recommendation, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/.

[45] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[46] ontoprise GmbH. *ObjectLogic Reference Guide*, 2010.

[47] ontoprise GmbH. *ObjectLogic Tutorial*, 2010.

[48] ontoprise GmbH. *OntoBroker Enterprise 6.0*, 2010.

[49] ontoprise GmbH. *OntoStudio 3.0*, 2010.

[50] A. D. Preece and R. Shinghal. Foundation and application of knowledge base verification. *International Journal of Intelligent Systems*, 9(8):683–701, 1994.

[51] Jörg Pührer, Stijn Heymans, and Thomas Eiter. Dealing with inconsistency when combining ontologies and rules using dl-programs. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *The Semantic Web: Research and Applications, Proceedings of the 7th Extended Semantic Web Conference (ESWC'10), Part I, Heraklion, Crete, Greece, May 30 - June 3, 2010*, volume 6088 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2010.

[52] Guilin Qi, Weiru Liu, and David A. Bell. A revision-based approach to handling inconsistency in Description Logics. *Artificial Intelligence Review*, 26(1-2):115–128, 2006.

[53] Guilin Qi and Fangkai Yang. A survey of revision approaches in Description Logics. In *Proceedings of the Twenty-First International Workshop on Description Logics (DL'08), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[54] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[55] Valdivino Santiago, Nandamudi L. Vijaykumar, and José Demisio Simões da Silva. Natural language requirements: Automating model-based testing and analysis of defects. In *Proceedings of the IX Workshop do Curso de Computaçãgo Aplicada, São José dos Campos, Brasil, October 21, 2009*. LIT/INPE, oct 2009.

[56] Stefan Schlobach. Diagnosing terminologies. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference (AAAI'05), July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 670–675. AAAI Press / The MIT Press, 2005.

[57] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of Description Logic terminologies. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco, Mexico, August 9-15, 2003*, pages 355–362. Morgan Kaufmann, 2003.

[58] Valentin Zacharias. *Tool Support for Finding and Preventing Faults in Rule Bases*. PhD thesis, Universität Karlsruhe, 2008.

[59] Xiaowang Zhang, Guohui Xiao, and Zuoquan Lin. A tableau algorithm for handling inconsistency in OWL. In Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, Elena Paslaru Bontas Simperl, Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Paslaru Bontas Simperl, editors, *Proceedings of the Sixth European Semantic Web Conference (ESWC'09), Heraklion, Crete, Greece, May 31-June 4, 2009*, volume 5554 of *Lecture Notes in Computer Science*, pages 399–413. Springer, 2009.

Glossary

ABox Synonym of Assertion Box, 3, 57

Assertion Box The Assertion Box is the population of assertions. Another way of looking at an assertion is to consider it as a fact. The Assertion box in OWL is restricted to unary and binary facts., 3, 57

Consistency In its narrower sense, consistency refers to the absence of contradictory information. A knowledge base is consistent if it has a model. Inconsistency, on the other hand, is the presence of a contradiction. A knowledge base is inconsistent if it has no model. In Task 2.4 - Consistency Maintainence, we refer to this notion of inconsistency as contradiction and also identify completeness, relevance, and language conformance as consistency related problem categories., 1, 3, 7, 15, 46, 57, 69, 71, 72, 75

Datalog Datalog is a query and rule language for deductive databases that syntactically is a subset of Prolog., 15

Decidability A decision problem, i.e., a yes-or-no question with a potentially infinite input domain, is decidable if there is an algorithm that computes the correct answer for every finite input within a finite amount of time., 63

Description Logics Description Logics (DLs) are a family of knowledge representation languages. The modeling primitives in most DLs are classes, which represent sets of objects, properties, which are relations between classes, and individuals. Constants may be defined using logical axioms. The language constructs available for writing such axioms depends on the DL at hand. Typical language constructs include class intersection, union, and complement, as well as universal and existential property restrictions., 57, 67

DL-Programs DL-Programs is a loosely coupled approach of integration of Ontology and Rules., 67

Entailment In model theory, a set S of formulas entails or implies a formula F if every model of S is also a model of F., 6, 63, 72

F-logic F-logic is a deductive, object oriented database language which combines the declarative semantics and expressiveness of deductive database languages with the rich data modeling capabilities supported by the object oriented data model. Classes and properties are represented as terms and interpreted as objects. This is one of the major differences to OWL where classes are represented as unary predicates and properties are represented as binary predicates. F-logic has been proposed for as an ontology language for the Semantic Web as well as extending existing approaches with F-logic based rules., 15

Knowledge engineer A knowledge engineer is a person who masters a knowledge acquisition methodology; a knowledge engineer need not have knowledge of any specific domain except the generic domain., 59, 61

Model A model of a sentence or a theory in the sense of model theory is a structure that satisfies the sentence, respectively theory. Intuitively, this means that the interpretation represented by the structure makes the sentence or theory true., 4, 5, 58–61, 68, 75–79, 81

ObjectLogic ObjectLogic is a newly developed ontology language which is based on the development of F-logic. It is developed at ontoprise GmbH., 15

OntoBroker 6.x OntoBroker http://www.ontoprise.de/en/home/products/ontobroker/ (cf. [AKL09], [Hey09]) is a Semantic Web middleware with inference engine that allows seamlessly accessing heterogeneous data sources using ontologies. OntoBroker 6.0 comes in different flavours. It incorporates an ObjectLogic engine (cf. [AKL09]) for ObjectLogic and RDF/S reasoning, and an OWL reasoner (cf. [dB09] section 2.2.2,) for reasoning over OWL ontologies. Within OntoBroker 6.1 it is planed to integrate OWL 2 RL reasoning into the ObjectLogic reasoner. The syntactical integration of OWL 2 into the ObjectLogic data model has already be done. The axiomatization of OWL 2 RL is already in progress (as by 22. Sep. 2010). There will be no separate reasoner for OWL., 20, 26, 32

OntoStudio 3.x OntoStudio http://www.ontoprise.de/en/home/products/ontostudio/ is a professional development environment for building, testing and maintaining ontologies. Data in various formats can be imported. OntoStudio is based on the eclipse framework. Therefore, OntoStudio has a modular design that is familiar to a large community of developers. Its rich set of features can be easily extended by writing new plug-ins that integrate via well-defined extension-points., 32

OWL The Web Ontology Language OWL is an ontology language for the Semantic Web that extends Description Logics. To RDFS it adds features such as class intersection, union and complement, local property restrictions, cardinality restrictions, and reflexive, symmetric, functional, transitive and inverse properties., 26–29, 49, 59, 62, 78

Reasoner A reasoner, also called a reasoning engine, is a software that can compute logical consequences of a given knowledge base. Many reasoners support fragments of first-order predicate logic. Reasoners for Description Logics include Pellet, OntoBroker, FaCT++, KAON2, and RacerPro., 3, 6, 59, 65, 78, 81

Rule In the context of ONTORULE, a rule is a statement expressing a fact or process that depends on a condition., 3, 15, 46, 67