Inconsistent Ontology Diagnosis: Framework and Prototype

Stefan Schlobach and Zhisheng Huang
(Vrije Universiteit Amsterdam)

Abstract.
EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D3.6.1(WP3.6)
In this document, we present a framework for inconsistent ontology diagnosis and repair by defining a number of new non-standard reasoning services to explain inconsistencies through pinpointing.
We developed two different types of algorithms for the framework, and we describe these algorithms in some detail. Both algorithms have been prototypically implemented as the DION (Debugger of Inconsistent ONtologies) and MUPSter system. The first implements a bottom-up approach to calculate pinpoints by the support of an external DL reasoner, the second using a specialised tableau-based calculus.
Keyword list: ontology management, inconsistency diagnosis, ontology reasoning

Document Id.	SEKT/2005/D3.6.1/v0.9
Project	SEKT EU-IST-2003-506826
Date	October 11, 2005
Distribution	internal

SEKT 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 IST-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE, UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana, Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP, UK
Tel: +44 114 222 1891, Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006 Madrid, Spain
Tel: +34 913 349 797, Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe, Germany
Tel: +49 721 50980912, Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam, The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Siemens Business Services GmbH & Co. OHG
Otto-Hahn-Ring 6
81739 Munich, Germany
Tel: +49 89 636 40 225, Fax: +49 89 636 40 233
Contact person: Dirk Ramhorst
E-mail: dirk.ramhorst@siemens.com

Empolis GmbH
Europaallee 10
67657 Kaiserslautern, Germany
Tel: +49 631 303 5540, Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe, Germany
Tel: +49 721 608 6592, Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Innsbruck
Institute of Computer Science
Technikerstraße 13
6020 Innsbruck, Austria
Tel: +43 512 507 6475, Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Kea-pro GmbH
Tal
6464 Springen, Switzerland
Tel: +41 41 879 00, Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Sirma Group Corp., Ontotext Lab
135 Tsarigradsko Shose
Sofia 1784, Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vallès)
Barcelona, Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Executive Summary

There are two main ways to deal with inconsistent ontologies. One is to simply avoid the inconsistency and to apply a non-standard reasoning method to obtain meaningful answers. Another approach is to diagnose and repair it when encountering inconsistencies. This document focus on the latter.

We present a framework for inconsistent ontology diagnosis and repair by defining a number of new non-standard reasoning services to explain inconsistencies through pinpointing. We also integrate the ideas of debugging in the context of model based diagnosis.

We developed two different types of algorithms for the framework. Both algorithms have been prototypically implemented as part of the research described by this document. The first, called DION (Debugger of Inconsistent ONtologies) implements a bottom-up approach to calculate pinpoints by the support of an external DL reasoner. The second one, called the MUPSter system uses a a specialised labelled tableau-based calculus for the same purpose. This documents describes both the algorithmic and the implementational aspects of the systems in some detail.

FvHTEST

Contents

Chapter 1

Introduction

Ontologies play a crucial role in the Semantic Web (SW), as they allow ”intelligent agents” to share information in a semantically unambiguous way, and to reuse domain knowledge (possibly created by external sources). However, this makes SW technology highly dependent of the quality, and, in particular, of the correctness of the applied ontology. Two general strategies for quality assurance are predominant, one based on developing more and more sophisticated ontology modelling tools, the second one based on logical reasoning. In this paper we will focus on the latter. With the advent of expressive ontology languages such as OWL and its close relation to Description Logics (DL), non-trivial implicit information, such as the `is-a` hierarchy of classes, can often be made explicit by logical reasoners. More crucially, however, state-of-the art DL reasoners can efficiently detect incoherences even in very large ontologies. The practical problem remains what to do in case an ontology has been detected to be incoherent.

Our work was motivated by the development of the DICE[1] terminology. DICE implements frame-based definitions of diagnostic information for the unambiguous and unified classification of patients in Intensive Care medicine. The representation of DICE is currently being migrated to an expressive Description Logic (henceforth DL) to facilitate logical inferences. Figure 1.1 shows an extract of the DICE terminology. In [7] the authors describe the migration process in more detail. The resulting DL terminology (usually called a “TBox”) contains axioms such as the following, where classes (like BODYPART) are translated as concepts, and slots (like REGION) as roles:

$$\begin{aligned} &\mathit{Brain} \dot{\sqsubseteq} \mathit{CNS} \sqcap \exists \mathit{systempart}.\mathit{NervousSystem} \sqcap \\ &\quad \mathit{BodyPart} \sqcap \exists \mathit{region}.\mathit{HeadAndNeck} \sqcap \forall \mathit{region}.\mathit{HeadAndNeck} \\ &\mathit{CNS} \dot{\sqsubseteq} \mathit{NervousSystem} \end{aligned}$$

Developing a coherent terminology is a time-consuming and error-prone process. DICE defines more than 2400 concepts and uses 45 relations. To illustrate some of the

[1] DICE stands for “Diagnoses for Intensive Care Evaluation”. The development of the DICE terminology has been supported by the NICE foundation.

CLASS	SUPERCLASS	SLOT	SLOT-VALUE
BRAIN	BODYPART CNS	REGION SYSTEM PART	HEAD AND NECK NERVOUS SYSTEM
CNS	NERVOUS SYSTEM		

Figure 1.1: An extract from the DICE terminology (frame-based).

problems, take the definition of a "brain" which is incorrectly specified, among others, as a "CNS" (central nervous-system) and "body-part" located in the head. This definition is contradictory as nervous-systems and body-parts are declared disjoint in DICE. Fortunately, current Description Logic reasoners, such as RACER [11] or FaCT [13], can detect this type of inconsistency and the knowledge engineer can identify the cause of the problem. Unfortunately, many other concepts are defined based on the erroneous definition of "brain" forcing each of them to be erroneous as well. In practice, DL reasoners provide lists of hundreds of unsatisfiable concepts for the DICE TBox and the debugging remains a jigsaw to be solved by human experts, with little additional explanation to support this process.

There are two main ways to deal with inconsistent ontologies. One is to simply avoid the inconsistency and to apply a non-standard reasoning method to obtain meaningful answers. In [14, 15], a framework of reasoning with inconsistent ontologies, in which pre-defined selection functions are used to deal with concept relevance, is presented The notion of "concept relevance" can be used for reasoning with inconsistent ontologies.

An alternative approach to deal with logical contradictions is to debug the ontology whenever a problem is encountered. In this document, we will focus on this *debugging* and *diagnosis* process. By *debugging* we understand the identification and elimination of modelling errors when detecting logical contradictions in a knowledge base. Debugging requires an *explanation* for the logical incorrectness and, as a second step, its *correction*. In this paper we will focus on the former as the latter requires an understanding of the meaning of represented concepts.

Our experience with debugging DICE provides some hands-on examples for the problem at hand: take the contradictory definition of brains in the DICE anatomy specification. What information is useful for correcting the knowledge base? First, we have to identify the precise position of errors within a TBox; that is, we need a procedure to single out the axioms causing the contradiction. The axioms for *Brain* and *CNS* form such a minimal incoherent subset of the DICE terminology. Formally, we introduce *minimal unsatisfiability-preserving sub-TBoxes* (abbreviated MUPS) and *minimal incoherence-preserving sub-TBoxes* (MIPS) as the smallest subsets of axioms of an incoherent terminology preserving unsatisfiability of a particular, respectively of at least one unsatisfiable concept. These notions will also be extended to full ontology inconsistency involving instances.

An orthogonal view on inconsistent ontologies is based on the traditional model-based *diagnosis* which has been studies over many years in the AI community[25]. Here the aim is to find minimal fixes, i.e. minimal subsets of an ontology that need to be repaired or removed to render an ontology logically correct, and therefor usable again.

To calculate pinpoints and diagnoses, there are basically two approaches, a bottom-up method using the support of an external reasoner, and a top-down implementation of a specialised algorithm. In this paper we describe one such approach each, the former based on the systematic enumerations of terminologies of increasing size based on selection functions on axioms, the latter on Boolean minimisation of labels in a labelled tableau calculus.

Both methods have been implemented as prototypes. The prototype for the informed bottom-up approach is called DION, which stands for a Debugger of Inconsistent ONtologies, the prototype of the specialised top-down method is called MUPSter. In this document, we are going to report on the implementation issue of both these systems, and provide a basic introduction on how to use the systems for debugging.

This document is organised as follows: Chapter 2 proposes a framework of inconsistent ontology diagnosis and repair called *pinpointing*. In Chapter 3 we introduce two frameworks of algorithms for pinpointing, a top-down approach based on the analysis of the properties of labelled tableau, and a bottom-up approach based on an informed enumeration of terminologies of increasing size. Chapter 4 describes the prototypes MUPSter and DION, which implement the two methods, before we conclude in Chapter 6.

Chapter 2

Inconsistent Ontologies: A Framework

This chapter deals with debugging and diagnosis of inconsistent Description Logic ontologies. Description Logics are a family of well-studied set-description languages which have been in use for over two decades to formalise knowledge. They have a well-defined model theoretic semantics, which allows for the automation of a number of reasoning services.

We shall not give a formal introduction into Description Logics here, but point to the second chapter of the DL handbook [2] for an excellent introduction. Briefly, in DL concepts will be interpreted as subsets of a domain, and roles as binary relations. In a terminological component $\mathcal{T}$ (called TBox) the interpretations of concepts can be restricted to the *models* of $\mathcal{T}$. Let, throughout the paper, $\mathcal{T} = \{Ax_1, \ldots, Ax_n\}$ be a set of (terminological) axioms, where Ax_i is of the form $C_i \dot{\sqsubseteq} D_i$ for each $1 \leq i \leq n$ and arbitrary concepts C_i and D_i.

Let $\mathcal{U}$ be a finite set, called the universe. A mapping $\mathcal{I}$, which interprets DL concepts as subsets of $\mathcal{U}$ is a *model* of a terminological axiom $C \dot{\sqsubseteq} D$, if, and only if, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A *model for a TBox* $\mathcal{T}$ is an interpretation which is a model for all axioms in $\mathcal{T}$. Based on this semantics a TBox can be checked for *incoherence*, i.e., whether there are *unsatisfiable* concepts: concepts which are necessarily interpreted as the empty set in all models of the TBox. More formally

1. A concept A is *unsatisfiable* w.r.t. a terminology $\mathcal{T}$ if, and only if, $A^{\mathcal{I}} = \varnothing$ for all models $\mathcal{I}$ of $\mathcal{T}$.

2. A terminology $\mathcal{T}$ is *incoherent* if there exists a concept-name in $\mathcal{T}$, which is unsatisfiable.

In an assertional component, the so-called *ABox*, properties of instances of the domain and relations between instances can be specified. Formally, an ABox is a set of *assertions* of the form $i : C$ and $r(i, j)$, where C and r are concepts and relations. and where i and j are individual names. An interpretation $\mathcal{I}$ is then a *model for an ABox* $\mathcal{A}$ if it is a model

for all assertions, i.e. if, and only if, $i^{\mathcal{I}} \in C^{\mathcal{I}}$ and $r^{\mathcal{I}}(i^{\mathcal{I}}, j^{\mathcal{I}})$ for $i : C \in \mathcal{A}$ and $r(i, j) \in \mathcal{A}$, respectively. Finally, a *Description Logic ontology* (henceforth simply *ontology*) is a pair $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, and a *model of this ontology* $\mathcal{O}$ is an interpretation which is both a model for $\mathcal{A}$ and $\mathcal{T}$. We will often refer to the set of terminological axioms and assertions simply as the *axioms* of the ontology.

Based on these definition we can now define a third logical property which points to a potential modelling error, inconsistency.

3. An ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is *inconsistent* if it has no models.

Conceptually, these three cases are all *simple* modelling errors because we assume that a knowledge modeller would not specify something an impossible concept in a complex way.

In this chapter we study ways of *explaining* incoherence and unsatisfiability in DL terminologies, and inconsistency of ontologies. We propose to simplify an ontology $\mathcal{O}$ in order to reduce the available information to the root of the logical error. More concretely we will exclude axioms which are irrelevant to the contradiction. We will call this process *pinpointing*.

2.1 Explaining Errors: Pinpointing

In this section we will formally introduce axiom pinpointing as a first step in explaining the logical contradictions of unsatisfiability of a concept, incoherence of a terminology and inconsistency of an ontology. By axiom pinpointing, we mean the identification of minimal subsets of a terminology or an ontology containing logical contradictions.

Let us be more precise: *axiom pinpointing* means identifying debugging-relevant axioms, where an axiom is *relevant* if a contradictory TBox becomes coherent once the axiom is removed or if, at least, a particular, previously unsatisfiable concept turns satisfiable. Consider the following (incoherent) TBox $\mathcal{T}^*$, where A, B and C are primitive and $A_1, \ldots, A_7$ defined concept names:

ax_1: $A_1 \dot{\sqsubseteq} \neg A \sqcap A_2 \sqcap A_3$	ax_2: $A_2 \dot{\sqsubseteq} A \sqcap A_4$
ax_3: $A_3 \dot{\sqsubseteq} A_4 \sqcap A_5$	ax_4: $A_4 \dot{\sqsubseteq} \forall s.B \sqcap C$
ax_5: $A_5 \dot{\sqsubseteq} \exists s.\neg B$	ax_6: $A_6 \dot{\sqsubseteq} A_1 \sqcup \exists r.(A_3 \sqcap \neg C \sqcap A_4)$
ax_7: $A_7 \dot{\sqsubseteq} A_4 \sqcap \exists s.\neg B$	

The set of unsatisfiable concept names that will be returned by a DL reasoner is $\{A_1, A_3, A_6, A_7\}$. Although this is still of manageable size, it hides crucial information, e.g., that unsatisfiability of A_1 depends on unsatisfiability of A_3, which is incoherent because of the contradictions between A_4 and A_5. We will use this example to explain our debugging methods.

Minimal unsatisfiability-preserving sub-TBoxes (MUPS)

MIPS are incoherent sets of terminological axioms which can, possibly, not be traced back to the unsatisfiability of a particular concept name. If we are interested in the causes for unsatisfiability of a particular concept we need a more fine-grained notion, called *minimal unsatisfiability-preserving sub-TBoxes* of a TBox and a particular concept.

Unsatisfiability-preserving sub-TBoxes of a TBox $\mathcal{T}$ and an unsatisfiable concept A are subsets of $\mathcal{T}$ in which A is unsatisfiable. In general there are several of these sub-TBoxes and we select the minimal ones, i.e., those containing only axioms that are necessary to preserve unsatisfiability.

Definition 1 Let A be a concept which is unsatisfiable in a TBox $\mathcal{T}$. A set $\mathcal{T}' \subseteq \mathcal{T}$ is a *minimal unsatisfiability-preserving sub-TBox (MUPS)* of $\mathcal{T}$ if

- A is unsatisfiable in $\mathcal{T}'$, and
- A is satisfiable in every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$.

We will abbreviate the set of MUPS of $\mathcal{T}$ and A by $mups(\mathcal{T}, A)$. MUPS for our example TBox $\mathcal{T}^*$ and its unsatisfiable concepts are:

$$\begin{aligned} mups(\mathcal{T}^*, A_1) &= \{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\} \\ mups(\mathcal{T}^*, A_3) &= \{\{ax_3, ax_4, ax_5\}\} \\ mups(\mathcal{T}^*, A_6) &= \{\{ax_1, ax_2, ax_4, ax_6\}, \\ &\qquad \{ax_1, ax_3, ax_4, ax_5, ax_6\}\} \\ mups(\mathcal{T}^*, A_7) &= \{\{ax_4, ax_7\}\} \end{aligned}$$

MUPS are useful for relating unsatisfiability to sets of axioms but we will also use them in Section 3.1.1 to calculate MIPS.

Minimal incoherence-preserving sub-TBoxes (MIPS)

MIPS are the smallest subsets of an original TBox preserving unsatisfiability of at least one atomic concept.

Definition 2 Let $\mathcal{T}$ be an incoherent TBox. A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a *minimal incoherence-preserving sub-TBox (MIPS) of* $\mathcal{T}$ if

- $\mathcal{T}'$ is incoherent, and
- every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$ is coherent.

We will abbreviate the set of MIPS of $\mathcal{T}$ by $mips(\mathcal{T})$. For $\mathcal{T}^*$ we get three MIPS:

$$mips(\mathcal{T}^*) \quad = \{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$$

It can easily be checked that each of the three incoherent TBoxes in $mips(\mathcal{T}^*)$ is indeed a MIPS as taking away a single axiom renders each of the three coherent. The first one signifies, for example, that the first two axioms are already contradictory without reference to any other axiom, which suggests a modelling error already in these two axioms.

We will refer to the explanation of unsatisfiability of concepts and incoherence of a terminology as terminological pinpointing. In a next step, we will have to tackle the case of inconsistency of a set of assertions with respect to a terminology.

Cores

Minimal incoherence-preserving sub-TBoxes and ontologies identify smallest sets of axioms causing the original ontology to be incoherent. In terminologies such as DICE, which are created through migration from other representation formalisms, there are several such sub-TBoxes, each corresponding to a particular contradictory terminology. *Cores* are now sets of axioms occurring in several of these incoherent TBoxes. The more MIPS such a core belongs to, the more likely its axioms will be the cause of contradictions.

Definition 3 Let $\mathcal{T}$ be a TBox. A non-empty intersection of n different MIPS in $mips(\mathcal{T})$ (with $n \geq 1$) is called a *MIPS-core of arity* n (or simply n*-ary core*) for $\mathcal{T}$.

Instead of *MIPS-cores of arity* n we will also talk of n*-ary cores*. Every set containing precisely one MIPS is, at least, a 1-ary core. The most interesting cores of a TBox, $\mathcal{T}$, are those with axioms that are present in as many MIPS of $\mathcal{T}$ as possible, i.e., having maximal arity. On the other hand, the size of a core is also significant, as a bigger size points to clusters of axioms causing contradictions in combination only.

In our running example, axiom ax_4 occurs both in $\{ax_3, ax_4, ax_5\}$ and $\{ax_4, ax_7\}$, which makes $\{ax_4\}$ a core of arity 2 for $\mathcal{T}_1$, which is the core of maximal arity in this example.

Minimal inconsistency preserving Sub-Ontologies (MISO)

Finally, we have to consider the case where a full ontology is inconsistent, i.e. where there is no model of all assertions with respect to the background terminology. The reason for this logical error can be twofold, in an assertion a counter-example to one or more terminological axioms is found, and the terminology has to be fixed. Alternatively, assertions could be erroneous. To cover both cases, we define here a simple unifying approach of calculating minimal inconsistent sub-ontologies, where we reduce both the terminology and the assertions.

Definition 4 Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be an inconsistent ontology. An ontology $\mathcal{O}' = (\mathcal{T}', \mathcal{A}')$ where $\mathcal{T}' \subseteq \mathcal{T}$ and $\mathcal{A}' \subseteq \mathcal{A}$ is a *minimal inconsistency preserving sub-ontology (MISO) of* $\mathcal{O}$ if, and only if,

- $\mathcal{O}'$ is inconsistent, and
- every sub-ontology $\mathcal{O}'' = (\mathcal{T}'', \mathcal{A}'')$, where $\mathcal{T}'' \subset \mathcal{T}'$ and $\mathcal{A}'' \subseteq \mathcal{A}'$ is coherent.

Note, that we presented this definition of MISOs purely on inconsistency, without any priory assumptions, such as for example to require the terminology to be coherent.

As an example, suppose that the following set of assertions $\mathcal{A}^*$ is given, and interpreted with respect to our example terminology $\mathcal{T}^*$.

$\mathcal{T}^* = \{$	ax_1:	$A_1 \dot{\sqsubseteq} \neg A \sqcap A_2 \sqcap A_3$	ax_2:	$A_2 \dot{\sqsubseteq} A \sqcap A_4$
	ax_3:	$A_3 \dot{\sqsubseteq} A_4 \sqcap A_5$	ax_4:	$A_4 \dot{\sqsubseteq} \forall s.B \sqcap C$
	ax_5:	$A_5 \dot{\sqsubseteq} \exists s.\neg B$	ax_6:	$A_6 \dot{\sqsubseteq} A_1 \sqcup \exists r.(A_3 \sqcap \neg C \sqcap A_4)$
	ax_7:	$A_7 \dot{\sqsubseteq} A_4 \sqcap \exists s.\neg B$		
$\mathcal{A}^* = \{$	ass_1:	$i_1{:}A_1$	ass_2:	$i_2{:}A_4$
	ass_3:	$i_3{:}\neg B$	ass_4:	$s(i_2, i_3)$
	ass_5:	$i_4{:}A_4 \sqcap A_5$	ass_6:	$i_2{:}\forall s.B$

There are a number of MISOs, for example,

$\mathcal{O}_1^* = (\{ax_1, ax_2\}, \{ass_1\}), \quad \mathcal{O}_2^* = (\{ax_4\}, \{ass_2, ass_3, ass_4\}),$
$\mathcal{O}_3^* = (\varnothing, \{ass_3, ass_4, ass_6\}), \quad \mathcal{O}_4^* = (\{ax_4, ax_5\}, \{ass_5\}).$

Some comments are in order: the most simple case is the ontology $\mathcal{O}_1^*$, which is inconsistent because A_1 is unsatisfiable in the terminology. In this case, the MISO consists of the assertion ass_1 which forces i_1 to be an instance of the unsatisfiable concept A_1, and the MUPS for A_1.

Unfortunately, inconsistency is not always due to unsatisfiability of concepts in the TBox, as the other example show. In $\mathcal{O}_2^*$, the axiom ax_4 forces every r successor to be in B, whereas the assertions ass_2, ass_3 and ass_4 force the particular role filler i_3 to be in $\neg B$.

In the third example, we show that inconsistency can even arise without a terminology involved, as the definition of an s relation between i_2 and i_3 and the universal quantification over s in ass_6 contradicts the assertion that $i_3 : \neg B$, even without any necessary background knowledge. Finally, there are cases, such as $\mathcal{O}_4^*$, when an individual is asserted to be an instance of two classes

Obviously, the notion of a core could be extended easily to MISOs.

Languages and Algorithms

The definitions for axiom pinpointing are language independent, as they are simply based on subsets of sets of axioms, where the structure of the individual axioms is not relevant. However, the calculation of explanations varies between languages, depending on the expressiveness of specialised algorithms (for our top-down methods) or of the general purpose reasoning tools such as RACER (for our bottom-up approach). In the following section 3.1.1 we will give a specialised (and complete) algorithm to calculate MIPS and MUPS in the case of unfoldable $\mathcal{ALC}$-TBoxes, and a general algorithm for expressive languages such as $\mathcal{SHIQ}$.

First, however, let us place our pinpointing method in the context of the well studied diagnosis framework of Reiter and de Kleer.

2.2 Suggesting Fixes: Model-based Diagnosis

The specific problem that is tackled in the theory of model-based diagnosis is to find minimal fixed to inconsistent systems, i.e., smallest sets of components, that have to be fixed in order to turn an inconsistent system into a coherent one. For the time being this means in our context to find minimal sets of axioms that one has to fix or remove to render a concept satisfiable, a TBox coherent, or an ontology consistent. This notion is orthogonal to pinpointing, as MIPS, MUPS and MISOs can be seen as special cases of conflict sets, and diagnoses can then be calculated from these conflict sets. We will introduce model-based diagnosis for logically contradicting ontologies by reducing the problem to the classical first-order representation of diagnosis in Reiter's original work.

In [25], Ray Reiter introduced a general framework for diagnosis based on first principles. He defines a *system* as a pair (Sd, Cmp) where Sd, the *system description*, is a set of first order (FO) sentences, and where Cmp, the *system components*, is a finite set of constants. To represent a terminology as a system for terminological debugging we represent satisfiability and incoherence as first-order satisfiability. Let $(C, x)^t$ be the standard translation from a concept C into FO-logic given a variable x, i.e. where, for example,

$$\begin{aligned}(C_1 \sqcap C_2, x)^t &= (C_1, x)^t \wedge (C_2, x)^t,\\ (\neg C, x)^t &= \neg (C, x)^t,\\ (\exists R.C, x)^t &= \exists R(x, y) \wedge (C, x)^t,\\ (A, x)^t &= A(x) \text{ for atomic concept names.}\end{aligned}$$

This standard translation can be trivially extended to TBox axioms: $(C \dot{\sqsubseteq} D)^t = \forall x.(C, x)^t \rightarrow (D, x)^t$. A TBox $\mathcal{T}$ translates into the first-order statement $\mathcal{T}^t = ax_1^t \wedge \ldots \wedge ax_n^t$.

Terminological and first-order satisfiability have a different flavour. Consider the translation of example TBox $\mathcal{T}^*$.

$$
\begin{array}{l}
\forall x(A_1(x) \rightarrow \neg A(x) \wedge A_2(x) \wedge A_3(x)) \wedge \\
\forall x(A_2(x) \rightarrow A(x) \wedge A_4(x)) \wedge \\
\ldots \\
\forall x(A_3(x) \rightarrow A_4(x)) \wedge A_5(x) \wedge \\
\forall x(A_4(x) \rightarrow \forall y(S(x,y) \rightarrow B(y)) \wedge C(x)) \wedge \\
\forall x(A_5(x) \rightarrow \exists y(S(x,y) \wedge \neg B(y))) \wedge \\
\forall x(A_6(x) \rightarrow A_1(x) \wedge \exists y(R(x,y) \wedge A_3(y) \wedge \neg C(y) \wedge A_4(y))) \wedge \\
\forall x(A_7(x) \rightarrow A_4(x) \wedge \exists y(S(x,y) \wedge \neg B(y)))
\end{array}
$$

which is consistent, even though A_1 is terminologically unsatisfiable. To make the translation satisfiability preserving, in the sense that a concept is satisfiable w.r.t. a TBox if, and only if, its first-order correspondence formula is satisfiable, we introduce *expectations*, such as $E = \{\exists y A(y)\}$. Then, A is satisfiable w.r.t. $\mathcal{T}$ if, and only if, $\mathcal{T}^t \cup E$ is satisfiable.[1]

To handle consistency of an ontology we do not need such a conceptual extension, because the standard translation of an ontology preserves logical consistency, in the sense that the translated ontology is first-order consistent if, and only if, the ontology is consistent w.r.t. the DL semantics. Here, the translation is a simple extension of the previous translation $(\cdot)^t$, where individual names are translated into first-order constants. Formally, we have the translation of the assertions defined by $(i : C)^t = (C, i)^t$ (using the previous definition for $(C, i)^t$, but now extended from variables to individuals) and $(R(i, j)^t = R(i, j)$. Let now $\mathcal{A}^t$ denote the translation of all the assertions in an ABox $\mathcal{A}$, then it is easy to see that $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is consistent if, and only if, $\mathcal{T}^t \wedge \mathcal{A}^t$ is consistent.

Terminological system descriptions

Terminological system description will capture the semantics of the terminology, and the components are those axioms, which are potentially erroneous. A distinct predicate $AB(\cdot)$ can be added to denote abnormality of components. [2] In our interpretation, truth of this predicate means that the axiom is erroneous, and should not contribute to the semantics of the terminology. The *(terminological) system description* $Sd(\mathcal{T})$ for a terminology $\mathcal{T}$ is the FO-formula:

$$\left(\neg AB(ax_1) \rightarrow ax_1^t\right) \wedge \ldots \wedge \left(\neg AB(ax_n) \rightarrow ax_n^t\right)$$

[1] The distinction between first-order and DL satisfiability is well-known, but has sometimes lead to confusion in discussions with people without the DL background.

[2] In this paper, we decided to take axioms as our components for a first diagnosis framework. The choice of components is one of the main control mechanisms for diagnostic purposes. We are currently investigating alternative sets of components, such as sub-concepts of concepts in axioms and assertions, but this ongoing research.

A diagnosis problem occurs when the terminological system description is unsatisfiable w.r.t. a set of expectations.[3]

Definition 5 Let $Sd(\mathcal{T})$ be terminological system description of $\mathcal{T}$, and Ex be a set of FO-formulas called *expectations*. Let, furthermore $Cmp \subseteq \mathcal{T}$ be a set of axioms, the *components*. We call $(Sd(\mathcal{T}), Ex, Cmp)$ a *(terminological) diagnosis problem* if $Sd(\mathcal{T}) \cup Ex)$ is inconsistent.

Let us look at two particular diagnosis problems, first, to explain unsatisfiability of a particular concepts, and,secondly, to explain incoherence. In what follows we will call the terminological diagnosis problem $(Sd(\mathcal{T}), \{\exists y A(y)\}, \mathcal{T})$ the *unsatisfiability problem*, and $(Sd(\mathcal{T}), \{\bigwedge_{A \in \mathcal{T}} \exists y A(y)\}, \mathcal{T})$ the *incoherence problem*.

Ontological system descriptions

To define an ontological diagnosis problem, we have to extend the notion of a terminological system description to ABoxes. Again, we take axioms and assertions as diagnostic components. The *(ontological) system description* $Sd(\mathcal{O})$ for an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{A} = \{ass_1, \ldots, ass_m\}$, is the FO-formula:

$$\big(Sd(\mathcal{T}) \wedge (\neg AB(ass_1) \rightarrow ass_1^t) \wedge \ldots \wedge (\neg AB(ass_1) \rightarrow ass_m^t)\big)$$

An ontological diagnosis problem occurs when the ontological system description is unsatisfiable.

Definition 6 Let $Sd(\mathcal{O})$ be ontological system description of $\mathcal{O}$. Let, furthermore $Cmp \subseteq \mathcal{T}$ be a set of axioms, the *components*. We call $(Sd(\mathcal{T}), Cmp)$ an *(ontological) diagnosis problem* if $Sd(\mathcal{T})$ is inconsistent.

Diagnosing terminologies and ontologies

We extend Reiter's definition of a diagnosis to terminological diagnosis problems by applying Proposition 3.4. of [25]. Note, that in this framework there is no conceptual difference for diagnosis of incoherence or unsatisfiability problems.

Definition 7 Let $\mathcal{T}$ be an incoherent terminology. A *(terminological) diagnosis* for $(Sd(\mathcal{T}), Ex, Cmp)$ is a minimal set $\Delta \subseteq \mathcal{T}$ such that

$$Sd(\mathcal{T}) \cup Ex \cup \{\neg AB(ax) \mid ax \in \mathcal{T} \setminus \Delta\} \text{ is consistent.}$$

[3] Expectations replace observations in Reiter's original framework. Observations follow from the semantics of the system.

In DL terms, a diagnosis Δ for the terminological diagnosis problem is a minimal sub-terminology of an unsatisfiable (or incoherent) terminology $\mathcal{T}$ (w.r.t. a concept A), such that A is unsatisfiable w.r.t. the remaining TBox $\mathcal{T} \setminus \Delta$ (respectively, that $\mathcal{T} \setminus \Delta$ is coherent).

From

$$mips(\mathcal{T}^*) = \{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$$

the 6 diagnoses

$$\begin{gathered}\{ax_1, ax_3, ax_7\},\\ \{ax_1, ax_4\},\\ \{ax_1, ax_5, ax_7\},\\ \{ax_2, ax_3, ax_7\},\\ \{ax_2, ax_5, ax_7\}\\ \{ax_2, ax_4\}\end{gathered}$$

can be derived. It can easily be checked, that for all these $\mathcal{T}^\Delta$, the TBoxes $\mathcal{T}' = \mathcal{T} \setminus \mathcal{T}^\Delta$ are coherent, and that there are no smaller $\mathcal{T}^\Delta$ with this property.

Finally, it remains to define diagnoses for the ontological inconsistency problem which is done in precisely the same way as before.

Definition 8 Let $\mathcal{O} = \{(\mathcal{T}, \mathcal{A})\}$ be an inconsistent ontology. An *(ontological) diagnosis* for $(Sd(\mathcal{O}), Cmp)$ is a minimal set $\Delta \subseteq (\mathcal{T} \cup \mathcal{A})$ such that

$$Sd(\mathcal{T}) \cup \{\neg AB(ax) \mid ax \in (\mathcal{T} \cup \mathcal{A}) \setminus \Delta\} \text{ is consistent.}$$

It should be noted that diagnoses and MIPS (MUPS) are complementary for debugging. A diagnosis suggests which axioms should be ignored (or fixed) to make the terminology coherent, but not every diagnosis necessarily contains the erroneous axiom. Suppose, that ax_2 and ax_4 contain errors, and that all other axioms are correct. The first diagnose $\{ax_1, ax_3, ax_7\}$, though correct, will not identify the error. In a large diagnoses space it might be difficult to find the right diagnosis. Each MIPS, on the other hand, definitively contains a culprit for the logical conflict.

Chapter 3

Algorithms

We present two general approaches to calculate explanations: a top-down method, which reduces the reasoning into smaller parts in order to explain a subproblem with reduced complexity, and an informed bottom-up approach, which enumerates possible solutions in a clever way. Both approaches will be represented for terminological reasoning only, but can, in principal, easily be extended to full ontology debugging.[1]

3.1 A Top-down Approach to Explanation

In order to calculate minimal incoherence preserving sub-terminologies (MIPS) we first calculate the minimal unsatisfiability preserving sub-terminologies (MUPS) for each unsatisfiable concept. This is done in a top-down way: we calculate the set of axioms needed to maintain a logical contradiction by expanding a logical tableau with labels. This method is efficient, as it requires a single logical calculation per unsatisfiable concept. On the other hand, it is based on a variation of a specialised logical algorithm, and only works for Description Logics for which such a specialised algorithm exists and is implemented. At the moment, such a purpose-build method only exists for the DL $\mathcal{ALC}$, and a restricted type of TBoxes, namely *unfoldable* ones.

3.1.1 Debugging Unfoldable ALC-TBoxes

Practical experience has shown that applying our methods on a simplified version of DICE can already provide valuable debugging information. We will therefore only provide algorithms for unfoldable $\mathcal{ALC}$-TBoxes [21] as this significantly improves both the computational properties and the readability of the algorithm.

[1] The extension of the bottom-up method is trivial, as we only have to define a new selection function and systematic enumeration. For the top-down approach things are a bit more complicated, as we now have analyse forests rather than trees. However, this seems to be a technical rather than a conceptual problem.

$(\sqcap)$:	**if**	$(a : C_1 \sqcap C_2)^{label} \in B$, but not both $(a : C_1)^{label} \in B$ and $(a : C_2)^{label} \in B$
	then	$B' := B \cup \{(a : C_1)^{label}, (a : C_2)^{label}\}$.
$(\sqcup)$:	**if**	$(a : C_1 \sqcup C_2)^{label} \in B$, but neither $(a : C_1)^{label} \in B$ nor $(a : C_2)^{label} \in B$
	then	$B' := B \cup \{(a : C_1)^{label}\}$ and $B'' := B \cup \{(a : C_2)^{label}\}$.
(Ax)	**if**	$(a : A)^{label} \in B$ and $(A \dot{\sqsubseteq} C) \in \mathcal{T}$
	then	$B' := B \cup \{(a : C)^{label \cup \{A \dot{\sqsubseteq} C\}}\}$.
$(\exists)$:	**if**	$(a : \exists R_i.C)^{label} \in B$, $R_i \in N_R$ and all other rules have been applied on all formulas over a, and if $\{(a : \forall R_i.C_1)^{label_1}, \ldots, (a : \forall R_i.C_n)^{label_n}\} \subseteq B$ is the set of universal formulas for a w.r.t. R_i in B,
	then	$B' := \{(b : C)^{label}, (b : C_1)^{label_1 \cup label}, \ldots, (b : C_n)^{label_n \cup label}\}$ where b is a new individual name not occurring in B.

Figure 3.1: Tableau Rules for $\mathcal{ALC}$-Satisfiability w.r.t. a TBox $\mathcal{T}$ (with Labels)

The calculation of MIPS depends on the MUPS only, and we will provide an algorithm to calculate these minimal unsatisfiability-preserving sub-TBoxes based on Boolean minimisation of terminological axioms needed to close a standard tableau ([2] Chapter 2).

Usually, unsatisfiability of a concept is detected with a fully saturated tableau (expanded with rules similar to those in Figure 3.1) where all branches contain a contradiction (or close, as we say). The information which axioms are relevant for the closure is contained in a simple label which is added to each formula in a branch. A *labelled formula* has the form $(a : C)^x$ where a is an individual name, C a concept and x a set of axioms, which we will refer to as *label*. A labelled branch is a set of labelled formulas and a tableau is a set of labelled branches. A formula can occur with different labels on the same branch. A branch is closed if it contains a clash, i.e. if there is at least one pair of formulas with contradictory atoms on the same individual. The notions of open branch and closed and open tableau are defined as usual and do not depend on the labels. We will always assume that any formula is in *negation normal form* (nnf) and newly created formulas are immediately transformed. We usually omit the prefix "labelled".

To calculate a minimal unsatisfiability-preserving TBox for a concept name A w.r.t. an unfoldable TBox $\mathcal{T}$ we construct a tableau from a branch B initially containing only $(a : A)^{\varnothing}$ (for a new individual name a) by applying the rules in Figure 3.1 as long as possible. The rules are standard $\mathcal{ALC}$-tableau rules with lazy unfolding, and have to be read as follows: assume that there is a tableau $T = \{B, B_1, \ldots, B_n\}$ with $n+1$ branches. Application of one of the rules on B yields the tableau $T' := \{B', B_1, \ldots, B_n\}$ for the $(\sqcap)$, $(\exists)$ and (Ax)-rule, $T'' := \{B', B'', B_1, \ldots, B_n\}$ in case of the $(\sqcup)$-rule.

Once no more rules can be applied, we know which atoms are needed to close a saturated branch and can construct a minimisation function for A and $\mathcal{T}$ according to the rules in Figure 3.2. A propositional formula ϕ is called a *minimisation function for A and $\mathcal{T}$* if A is unsatisfiable in every subset of $\mathcal{T}$ containing the axioms which are true in an assignment making ϕ true. In our case axioms are used as propositional variables in ϕ. As we can identify unsatisfiability of A w.r.t. a set S of axioms with a closed tableau

if *rule* = ($\sqcap$) has been applied to $(a : C_1 \sqcap C_2)^{label}$ and B' is the new branch
  return *min_function*$(a, B', \mathcal{T})$*;*
if *rule* = ($\sqcup$) has been applied to $(a : C_1 \sqcup C_2)^{label}$ and B' and B'' are the new branches
  return *min_function*$(a, B', \mathcal{T}) \wedge$ *min_function*$(a, B'', \mathcal{T})$*;*
if *rule* = ($\exists$) has been applied to $(a : \exists R.C)^{label}$, B' is the new branch and b the new variable
  return *min_function*$(a, B', \mathcal{T}) \vee$ *min_function*$(b, B', \mathcal{T})$*;*
if *rule* = (Ax) has been applied and B' is the new branch
  return *min_function*$(a, B', \mathcal{T})$*;*
if no further rule can be applied
  return: $\bigvee_{(a:A)^x \in B, (a:\neg A)^y \in B} (\bigwedge_{ax \in x} ax \wedge \bigwedge_{ax \in y} ax)$;

Figure 3.2: *min_function*$(a, B, \mathcal{T})$: Minimisation-function for the MUPS-problem

using only the axioms in S for unfolding, branching on a disjunctive rule implies that we need to join the functions of the appropriate sub-branches conjunctively. If an existential rule has been applied, the new branch B' might not necessarily be closed on formulas for both individuals. Assume that B' closes on the individual a but not on b. In this case *min_function*$(a, B, \mathcal{T}) = \bot$, which means that the related disjunct does not influence the calculation of the minimal incoherent TBox.

Based on the minimisation function *min_function*$(a, \{(a : A)^{\varnothing}\}, \mathcal{T})$ (let us call it ϕ) which we calculated using the rules in Figure 3.2 we can now calculate the MUPS for A w.r.t. $\mathcal{T}$. The idea is to use prime implicants of ϕ. A prime implicant $ax_1 \wedge \ldots \wedge ax_n$ is the smallest conjunction of literals[2] implying ϕ [24]. As ϕ is a minimisation function every implicant of ϕ must be a minimisation function as well and therefore also the prime implicant. But this implies that the concept A must be unsatisfiable w.r.t. the set of axioms $\{ax_1, \ldots, ax_n\}$. As $ax_1 \wedge \ldots \wedge ax_n$ is the smallest implicant we also know that $\{ax_1, \ldots, ax_n\}$ must be minimal, i.e. a MUPS. Theorem 3.1.1 captures this result formally.

Theorem 3.1.1 *Let A be a concept name, which is unsatisfiable w.r.t. an unfoldable $\mathcal{ALC}$-TBox $\mathcal{T}$. The set of prime implicants of the minimisation function min_function$(a, \{(a : A)^{\varnothing}\}, \mathcal{T})$ is the set $mups(\mathcal{T}, A)$ of minimal unsatisfiability-preserving sub-TBoxes of A and $\mathcal{T}$.*

PROOF. We first prove the claim that the propositional formula $\phi :=$ *min_function*$(a, \{(a : A)^{\varnothing}\}, \mathcal{T})$ is indeed a minimisation function for the MUPS problem w.r.t. an unsatisfiable concept A and a TBox $\mathcal{T}$. We show that a tableau starting on a single branch $B := \{(a : A)^{\varnothing}\}$ closes on all branches by unfolding axioms only, that are evaluated as true in an assignment making ϕ true. This saturated tableau Tab^* is a particular sub-tableau of the original saturated tableau Tab which we used to calculate

[2]Note that in our case all literals are non-negated axioms.

$min\_function(a, \{(a : A)^{\varnothing}\}, \mathcal{T})$, and it is this connection that we make use of to prove our first claim. Every branch in the new tableau is a subset of a branch occurring in the original one and we define *visible formulas* as those labelled formulas occurring in both tableaux. By induction over the rules applied to saturate Tab we can then show that each branch in the original tableau closes on at least one pair of visible formulas. If A is unsatisfiable w.r.t. $\mathcal{T}$, the tableau starting with the branch $\{(a : A)^{\varnothing}\}$ closes w.r.t. $\mathcal{T}$. As we have shown that this tableau closes w.r.t. $\mathcal{T}$ on visible formulas, it follows that Tab^* is closed on all branches, which proves the first claim. By another induction over the application of the rules in Figure 3.2 we can prove that ϕ is a *maximal* minimisation function, which means that $\psi \rightarrow \phi$ for every minimisation function ψ. This proves the first part of the proof; the first claim (and the argument from above) implies that every implicant of a minimisation function identifies an unsatisfiability-preserving TBox, and maximality implies that prime implicants identify the minimal ones.

To show that the conjunction of every MUPS $\{ax_1, \ldots, ax_n\}$ is a prime implicant of *min_function*$(a, \{(a : A)^{\varnothing}\}, \mathcal{T})$ is trivial as $ax_1 \wedge \ldots \wedge ax_n$ is a minimisation function by definition. But as we know that *min_function*$(a, \{(a : A)^{\varnothing}\}, \mathcal{T})$ is maximal we know that $ax_1 \wedge \ldots \wedge ax_n \rightarrow$*min_function*$(a, \{(a : A)^{\varnothing}\}, \mathcal{T})$ which implies that $ax_1 \wedge \ldots \wedge ax_n$ must be prime as otherwise $\{ax_1, \ldots, ax_n\}$ would not be minimal. □ Satisfiability

in $\mathcal{ALC}$ is PSPACE-complete, and calculating MUPS does not increase the complexity as we can construct the minimisation function in a depth-first way, allowing us to keep only one single branch in memory at a time. However, we calculate prime implicants of a minimisation function the size of which can be exponential in the number of axioms in the TBox. Therefore, approximation methods have to be considered in practice avoiding the construction of fully saturated tableaux in order to reduce the size of the minimisation functions.

From MUPS we can easily calculate MIPS, but we need an additional operation on sets of TBoxes, called *subset-reduction*. Let $M = \{\mathcal{T}_1, \ldots, \mathcal{T}_m\}$ be a set of TBoxes. The *subset-reduction* of M is the smallest subset $sr(M) \subseteq M$ such that for all $\mathcal{T} \in M$ there is a set $\mathcal{T}' \in sr(M)$ such that $\mathcal{T}' \subseteq \mathcal{T}$. A simple algorithm for the calculation of MIPS for $\mathcal{T}$ now simply follows from Theorem 3.1.2, which is a direct consequence of the definitions of MIPS and MUPS.

Theorem 3.1.2 *Let $\mathcal{T}$ be an incoherent TBox with unsatisfiable concepts $\Delta^{\mathcal{T}}$. Then, $mips(\mathcal{T}) = sr(\bigcup_{A \in \Delta^{\mathcal{T}}} mups(\mathcal{T}, A))$.*

Checking elements of $mips(\mathcal{T})$ for cores of maximal arity requires exponentially many checks in the size of $mips(\mathcal{T})$. In practice, we therefore apply a bottom-up method searching for maximal cores of increasing size stopping once the arity of the cores is smaller than 2.

3.2 An Informed Bottom-up Approach to Explanation

3.2.1 General Idea

In this section we propose an informed bottom-up approach to calculate MUPS by the support of an external DL reasoner, like RACER. The main advantage of this approach is that it can deal with any DL-based ontology if it has been supported by an external reasoner. Currently there exist several well-known DL reasoners, like RACER and FACT++. Those external DL reasoners have been proved to be very reliable and stable. They already support various DL-based ontology languages, including OWL. Thus, by the bottom-up approach we can obtain an OWL debugger almost for free, although the price is paid for its performance.

Given an unsatisfiable concept c and a formula set(i.e., an ontology) $\mathcal{T}$, we can calculate the MUPS of c by selecting a minimal subset Σ of $\mathcal{T}$ in which c is unsatisfiable in Σ. We use a similar selecting procedure which has been used in the system PION for reasoning with inconsistent ontologies[14]. In the PION approach, a selection function is designed to one which can extend selected subset by checking on axioms which are relevant to the current selected subset which starts initially with a query. Although the approach which is based this kind of relevance extension procedure may not give us the complete solution set of MUPS/MIPS, it is good enough to provide us an efficient approach for debugging inconsistent ontologies. We are going to report the evaluation of this informed bottom-up approach in the SEKT deliverable D3.6.2 entitled "Evaluation of Inconsistent Ontology Diagnosis".

3.2.2 Selection Function and Relevance Measure

Given an ontology (i.e., a formula set) Σ and a query ϕ, a selection function s is one which returns a subset of Σ at the step $k > 0$. Let $\mathbf{L}$ be the ontology language, which is denoted as a formula set. We have the general definition about selection functions as follows:

Definition 3.2.1 (Selection Functions) *A selection function s is a mapping $s : \mathcal{P}(\mathbf{L}) \times \mathbf{L} \times N \rightarrow \mathcal{P}(\mathbf{L})$ such that $s(\Sigma, \phi, k) \subseteq \Sigma$.*

In this approach, we extend the definition of the selection function so that it starts from a concept c instead of from a query (i.e., a formula ϕ). As we have discussed above, select functions are usually defined by a relevance relation between a formula and a formula set. We will use a relevance relation as the informed message to guide the search strategy for MUPS.

Definition 3.2.2 (Direct Relevance Relation) *A direction relevance relation $\mathcal{R}$ is a set of formula pairs. Namely, $\mathcal{R} \subseteq \mathbf{L} \times \mathbf{L}$.*

Definition 3.2.3 (Direct Relevance Relation between a Formula and a Formula Set) *Given a direction relevance relation $\mathcal{R}$, we can extend it to a relation $\mathcal{R}^+$ on a formula and a formula set, i.e., $\mathcal{R}^+ \subseteq \mathbf{L} \times \mathcal{P}(\mathbf{L})$ as follows:*

$$\langle \phi, \Sigma \rangle \in \mathcal{R}^+ \textit{iff there exists a formula } \psi \in \Sigma \textit{ such that } \langle \phi, \psi \rangle \in \mathcal{R}.$$

We have implemented the prototype of the informed bottom-up approach. The prototype is called DION, which stands for a Debugger of Inconsistent Ontologies. We will discuss the implementation of DION in Chapter 4. DION uses a DIG data format as its internal data representation format. Therefore, in the following, we define a direct relevance relation which is based on the ontology language DIG.

In DIG, concept axioms has only the following three forms: $impliesc(C_1, C_2)$, $equalc(C_1, C_2)$, and $disjoint(C_1, \cdots, C_n)$, which corresponds with the concept implication statement, the concept equivalence statement, and the concept disjoint statement respectively.

Given a formula ϕ, we use $C(\phi)$ to denote the set of concept names that appear in the formula ϕ.

Definition 3.2.4 (Direct concept-relevance) *An axiom ϕ is directly concept-relevant to a formula ψ, written $SynConRel(\phi, \psi)$, iff*
(i) $C_1 \in C(\psi)$ if the formula ϕ has the form $impliesc(C_1, C_2)$,
(ii) $C_1 \in C(\psi)$ or $C_2 \in C(\psi)$ if the formula ϕ has the form $equalc(C_1, C_2)$,
(iii) $C_1 \in C(\psi)$ or $\cdots$ or $C_n \in C(\psi)$ if the formula ϕ has the form $disjoint(C_1, \cdots, C_n)$.

Definition 3.2.5 (Direct concept-relevance to a set) *A formula ϕ is concept-relevant to a formula set Σ iff there exists a formula $\psi \in \Sigma$ such that ϕ and ψ are directly concept-relevant.*

For a terminology $\mathcal{T}$ and a concept c, we can define a selection function s in terms of the direct concept relevance as follows:

Definition 3.2.6 (Selection function on concept relevance)
(i) $s(\mathcal{T}, c, 0) = \{\psi \mid \psi \in \mathcal{T}$ and ψ is directly concept-relevant to $c\}$;
(ii)$s(\mathcal{T}, c, k) = \{\psi \mid \psi \in \mathcal{T}$ and ψ is directly concept-relevant to $s(\mathcal{T}, c, k-1)\}$ for $k > 0$.

In order to do so, we extend the definition of direct concept relevance so that we can say something like an axiom ψ is directly concept-relevant to a concept c, i.e., $SynConRel(\psi, c)$. It is easy to see that it does not change the definition of direct relevance relation.

Figure 3.3: Algorithm for $mups(\mathcal{T}, c)$

$k := 0$
$mups(\mathcal{T}, c) := \emptyset$
repeat
  $k := k + 1$
until c unsatisfiable in $s(\mathcal{T}, c, k)$
$\Sigma := s(\mathcal{T}, c, k) - s(\mathcal{T}, c, k-1)$
for all $\Sigma' \in \mathcal{P}(\Sigma)$ **do**
  for all $\phi \in \Sigma/\Sigma'$ **do**
    if $\Sigma' \cup \{\phi\} \notin mups(\mathcal{T}, c)$ **then**
      $\Sigma'' := s(\mathcal{T}, c, k-1) \cup \Sigma'$
      if c satisfiable in Σ'' and c unsatisfiable in $\Sigma'' \cup \phi$ **then**
        $mups(\mathcal{T}, c) := mups(\mathcal{T}, c) \cup \{\Sigma'' \cup \{\phi\}\}$
      end if
    end if
  end for
end for
$mups(\mathcal{T}, c) := \mathbf{MinimalityChecking}(mups(\mathcal{T}, c))$
return $mups(\mathcal{T}, c)$

Figure 3.4: Algorithm for the minimality checking on $mups(\mathcal{T}, c)$

for all $\Sigma \in mups(\mathcal{T}, c)$ **do**
  $\Sigma' := \Sigma$
  for all $\phi \in \Sigma'$ **do**
    if c unsatisfiable in $\Sigma' - \{\phi\}$ **then**
      $\Sigma' := \Sigma' - \{\phi\}$
    end if
  end for
  $mups(\mathcal{T}, c) := mups(\mathcal{T}, c)/\{\Sigma\} \cup \{\Sigma'\}$
end for
return $mups(\mathcal{T}, c)$

3.2.3 Algorithms

We use an informed bottom-up approach to obtain MUPS. In logics and computer science, an increment-reduction strategy is usually used to find minimal inconsistent sets[8]. Under this approach, the algorithm first finds a set of inconsistent sets, then reduces the redundant axioms from the subsets. Similarly, a heuristic procedure for finding MUPS consists of the following three stages:

- **Heuristic Extension**: Using a relevance-based selection function to find two subsets Σ and S such that a concept c is satisfiable in S and unsatisfiable in $S \cup \Sigma$.
- **Enumeration Processing**: Enumerating subsets S' of S to obtain a set $S' \cup \Sigma$ in which the concept c is unsatisfiable. We call those sets *c-unsatisfiable sets*.
- **Minimality Checking**: Reducing redundant axioms from those c-unsatisfiable sets to get MUPSs.

The following is an algorithm for MUPSs. The algorithm first finds two subsets Σ and S of $\mathcal{T}$. Compared with $\mathcal{T}$, the set Σ is relatively small. The algorithm then tries to exhaust the powerset of Σ to get c-unsatisfiable sets. Finally, by the minimality checking it obtains the MUPSs. We can define the minimality checking as a sub-procedure as shown in the algorithm 3.4.

The complexity of the algorithm 3.3 is exponent to $|\Sigma|$. Although Σ is much smaller than $\mathcal{T}$, it is still not very useful in the implementation. One of the improvement is to do pruning. We can check the subsets of Σ with increasing cardinality of the subsets. Namely, we can always pick up the subsets with a less cardinality first, (i.e., the power set of Σ is sorted). First, set the cardinality $n = 1$, namely pick up only one axiom ϕ in Σ, check if c is unsatisfiable in $\{\phi\} \cup s(\mathcal{T}, c, k-1)$. If 'yes', then it ignores any superset S such that $\{\phi\} \subset S$. After all of the subsets with the cardinality n have been checked, increase n by 1. Moreover, we can do checking and pruning during the powerset is built. That leads to the algorithm 3.5 in which we use the set S to book the c-satisfiable subsets.

Proposition 3.2.1 (Soundness of the Algorithms MUPS) *The algorithms to compute MUPS above are sound. In other words, they always return MUPSs.*

PROOF. It is easy to see that the concept c is always unsatisfiable for any element S in the set $mups(\mathcal{T}, c)$. Otherwise it is never added into the set. The minimality condition is achieved by the procedure of the minimality checking. Therefore, the algorithms for MUPSs are sound. □

Take our running example. To calculate $mups(\mathcal{T}_1, A_1)$, the algorithm first gets the set $\Sigma = \{ax_2\} = \{ax_1, ax_2\} - \{ax_1\}$. Thus, $mups(\mathcal{T}_1, A_1) = \{\{ax_1, ax_2\}\}$. We

Figure 3.5: Algorithm for $mups(\mathcal{T}, c)$ with pruning

$k := 0$
$mups(\mathcal{T}, c) := \emptyset$
repeat
  $k := k + 1$
until c unsatisfiable in $s(\mathcal{T}, c, k)$
$\Sigma := s(\mathcal{T}, c, k) - s(\mathcal{T}, c, k-1)$
$S := \{s(\mathcal{T}, c, k-1)\}$
for all $\phi \in \Sigma$ **do**
  for all $S' \in S$ **do**
    if c satisfiable in $S' \cup \{\phi\}$ and $S' \cup \{\phi\} \notin S$ **then**
      $S := S \cup \{S' \cup \{\phi\}\}$
    end if
    if c unsatisfiable in $S' \cup \{\phi\}$ and $S' \cup \{\phi\} \notin mups(\mathcal{T}, c)$ **then**
      $mups(\mathcal{T}, c) := mups(\mathcal{T}, c) \cup \{S' \cup \{\phi\}\}$
    end if
  end for
end for
$mups(\mathcal{T}, c) := \mathbf{MinimalityChecking}(mups(\mathcal{T}, c))$
return $mups(\mathcal{T}, c)$

Figure 3.6: Finding a MUPS for $\mathcal{T}$ in which a concept c is unsatisfiable

$\Sigma := \emptyset$
repeat
  for all $\phi_1 \in \mathcal{T} \setminus \Sigma$ **do**
    if $SynConcRel(\phi_1, c)$ or there is a $\phi_2 \in \Sigma$ such that $SynConRel(\phi_1, \phi_2)$ **then**
      $\Sigma := \Sigma \cup \{\phi_1\}$
    end if
  end for
until c is unsatisfiable in Σ
for all $\phi \in \Sigma$ **do**
  if c is unsatisfiable in $\Sigma - \{\phi\}$ **then**
    $\Sigma := \Sigma - \{\phi\}$
  end if
end for

can see that the algorithm cannot find that $S_1 = \{ax_1, ax_3, ax_4, ax_5\} \in mups(\mathcal{T}_1, A_1)$. However, it does not change the result of MIPS, because we have a subset $\{ax_3, ax_4, ax_5\}$ of S_1, which will be in $mups(\mathcal{T}_1, A_3)$. To calculate $mups(\mathcal{T}_1, A_6)$, the algorithm first gets the set $\Sigma = \{ax_2, ax_5\} = \{ax_1, ax_2, ax_3, ax_4, ax_5, ax_6\} - \{ax_1, ax_3, ax_4, ax_6\}$. Thus, $mups(\mathcal{T}_1, A_6) = \{\{ax_1, ax_3, ax_4, ax_5, ax_6\}\}$. Again, the algorithm cannot find that $\{ax_1, ax_2, ax_4, ax_6\} \in mups(\mathcal{T}_1, A_6)$. However, it does not affect MIPS, because $\{ax_1, ax_2\} \in mups(\mathcal{T}_1, A_1)$. Therefore, the algorithms for MUPSs proposed above are sound, but not complete. As we have argued above, this informed bottom-up approach is efficient for inconsistent ontology diagnosis.

Sometimes it is useful to find just a single MUPS by using the relevance relation without referring to a selection function. Algorithm 3.6 uses the increment-reduction strategy to find a single MUPS for an unsatisfiable concept, without a selection function. The algorithm finds a subset of the ontology in which the concept is unsatisfiable first, then reduces the redundant axioms from the subset.

We can obtain MUPSs for all unsatisfiable concepts. Based on those MUPSs, we can calculate MIPS, core, and pinpoints further, by using the standard algorithms which have been discussed in the previous chapter. Those data can be used for knowledge workers to repair the ontology to avoid unsatisfiable concepts[27, 26].

3.3 Calculating terminological diagnoses

Terminological diagnosis, as defined in the previous section, is an instance Reiter's diagnosis from first principles. Therefore, we can use Reiter's algorithms to calculate terminological diagnoses. What is required is a method to produce conflict sets, and we will discuss three different options for this. Let us first recall the basic methodology from [25].

Given an incoherent terminology $\mathcal{T}$, a *conflict set* for $(Sd(\mathcal{T}), Ex, Cmp)$ is a set $CS \subseteq Cmp$, such that $Sd(\mathcal{T}) \cup Ex \cup \bigcup_{ax \in CS} \{\neg AB(ax)\}$ is inconsistent. A conflict set is minimal if, and only of, no proper subset of it is a conflict set for the same diagnosis problem.

The following proposition is the basis for calculating diagnosis on the basis of conflict sets.

Proposition 3.3.1 ([25] Proposition 4.2.) A set $\Delta \subseteq \mathcal{T}$ is a diagnosis for a terminological diagnosis problem $(Sd(\mathcal{T}), Ex, Cmp)$ iff Δ is a minimal set such that $\mathcal{T} \setminus \Delta$ is not a conflict set of $(Sd(\mathcal{T}), Ex, Cmp)$.

The basic idea to calculate diagnosis from conflict sets is based on minimal hitting sets. Suppose C is a collection of sets. A *hitting set* for C is a set $H \subseteq \bigcup_{S \in C} S$ such that $H \cap S \neq \varnothing$ for each $S \in C$. A hitting set is minimal for C iff no proper subset of it is a hitting set for C.

This gives the basis of Reiter approach to calculate diagnoses given the following theorem which is a direct consequence of Corollary 4.5 in [25].

Theorem 3.3.1 A set $\Delta \in \mathcal{T}$ is a diagnosis for a terminological diagnosis problem $(Sd(\mathcal{T}), Ex, Cmp)$ iff Δ is a minimal hitting set for the collection of conflict sets for $(Sd(\mathcal{T}), Ex, Cmp)$.

To calculate minimal hitting trees Reiter introduces hitting set trees (HS-trees). For a collection C of sets, a HS-tree T is the smallest edge-labelled and node-labelled tree, such that the root is labelled by ✓ if C is empty. Otherwise it is labelled with any set in C. For each node n in T, let $H(n)$ be the set of edge labels on the path in T from the root to n. The label for n is any set $S \in C$ such that $S \cap H(n) = \varnothing$, if such a set exists. If n is labelled by a set S, then for each $\sigma \in S$, n has a successor, n_σ joined to n by an edge labelled by σ. For any node labelled by ✓, $H(n)$, i.e. the labels of its path from the root, is a hitting set for C.

Figure 3.7 shows a HS-tree T for the collection

$$C = \{\{1,2,3,4,5,6\}\{3,4,5\}, \{1,2,4,6\}, \{1,2\}, \{4,7\}\}$$

of sets. T is created breadth first, starting with root node n_0 labelled with $\{1,2,3,4,5,6\}$. For diagnostic problems the sets in the collection are conflict sets which are created on demand. In our case, conflict sets for a terminological diagnosis problem can be calculated by a standard DL engine because of the following simple proposition.

Proposition 3.3.2 For any set C of components (terminological axioms) in a terminological diagnostic problem, the FO-formula $Sd(\mathcal{T}) \cup Ex \cup \bigcup_{ax \in C} \{\neg AB(ax)\}$ is inconsistent if, and only if, A is unsatisfiable in $\mathcal{T} \setminus C$.

These calls are computationally expensive, which means that we have to minimise them. In Figure 3.7, those nodes are boxed, for which labels were created by calls to the prover. T reuses already calculated and smallest possible labels, and is pruned in a variety of ways, which are defined in detail in [25]. Just for example, node n_0 is relabelled with a subset $\{3,4,5\}$ of its label. We denote by $1^\times$, that element 1 is deleted. Note, that no successor for this element has to be created. Node n_6 has been automatically labelled with $\{4,7\}$, because the intersection of its path $h(n_6) = \{2,3\}$ is empty with an already existing conflict set in the tree.

3.3.1 Three ways of implementing diagnosis

The generality of Reiter's algorithm has the advantage of giving some levy for particular methodological choices. We implemented three ways of calculating conflict sets.

1. Use an optimised DL reasoner to return a conflict set in each step of the creation of the HS-tree. The only way to get conflict sets for an incoherent TBox $\mathcal{T}$ is to return $\mathcal{T}$ itself, i.e. the *maximal conflict set*.

2. Use an adapted DL reasoner to return *small conflict sets*, which it can derive from the clashes in a tableau proof.

3. Use a specialised method to return *minimal conflict sets*, e.g., using the methods of [27].

Diagnosis with maximal conflict sets

The most general way to calculate terminological diagnosis based on hitting sets is to use one of the state-of-the-art optimised DL reasoner. The advantage is obvious: the expressiveness of the diagnosis is only restricted by the expressiveness of the DL reasoning implemented in the reasoner. We use RACER, which allows to diagnose incoherent terminologies up to $\mathcal{SHIQ}$ without restriction on the structure of the TBox. The algorithm to use RACER is simple: if $\mathcal{T}$ is incoherent, return $\mathcal{T}$, other return $\varnothing$. As RACER is highly optimised we can expect to get the maximal conflict sets efficiently.

The disadvantage of this naive approach is that the conflict sets are huge, and even with reusing of node labels and pruning, the HS-tree become quickly to large to handle. Take TBox $\mathcal{T}^*$ with its incoherence problem $(Sd(\mathcal{T}^*), \{\bigwedge_{A\in\mathcal{T}} \exists y A(y)\}, \mathcal{T}^*, \varnothing)$, where related HS-tree already has 380 nodes, and needs 67 calls to RACER. We will see that the price we pay for the gain in expressiveness is too high, and that the smaller conflict sets are required.

Diagnosis with small conflict sets

The disadvantage of using a DL reasoner as a black-box is that they do not provide any information on which components contribute to the incoherence. Technically, this means which axioms contribute to the closure of the tableau. To show that already straightforward collecting of clash-enforcing axioms can dramatically improve the efficiency of diagnosis, we implemented a simple tableau calculus for unfoldable $\mathcal{ALC}$ TBoxes. This reasoner returns an unordered, and not necessarily minimal, list of axioms which are (indirectly) responsible for the clashes in the tableau. The basic idea is to label each formula with a set of axioms, which are added to a formula in the tableau whenever they are used to "unfold" a defined concept. This algorithm is not optimised, but returns small conflict sets, and the sizes of the HS-Trees decrease dramatically. Figure 3.7 shows the hitting tree for the incoherence problem for $\mathcal{T}^*$ where small conflict sets have been collected from tableau proofs. Compared to the previous method, there were only 14 nodes created, and 11 calls to the DL reasoner necessary.

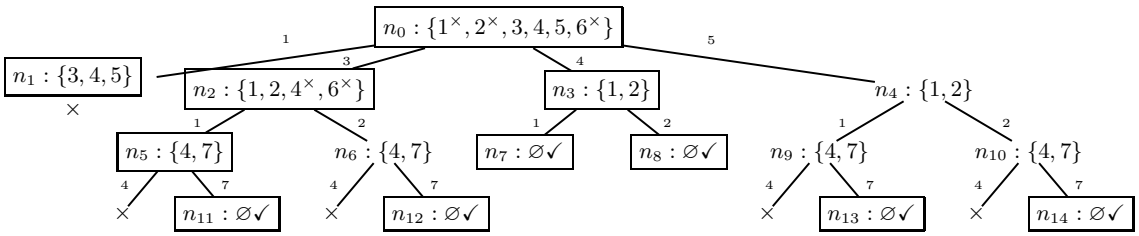

Figure 3.7: HS-Tree with small conflict sets

Diagnosis with minimal conflict sets

Previously, we recalled the notion of minimal unsatisfiability (and incoherence) preserving sub-terminologies MUPS and MIPS, which were introduced in [27] for the debugging of terminologies.

The MUPS of an incoherent terminology $\mathcal{T}$ and an unsatisfiable concept A are the minimal conflict sets for the unsatisfiability problem $(Sd(\mathcal{T}), \{\exists y A(y)\}, \mathcal{T}, \varnothing)$. It is easily checked that each MUPS $\{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\}$ for A_1 and $\mathcal{T}^*$ is indeed a minimal conflict set for the unsatisfiability problems $(Sd(\mathcal{T}^*), \exists y A_1(y)), \mathcal{T}^*, \varnothing)$. Based on the MUPS, it is straightforward to calculate MIPS, which are the conflict sets for the incoherence problem.

Proposition 3.3.3 The MIPS of an incoherent terminology $\mathcal{T}$ are the minimal conflict sets for the incoherence problem $(Sd(\mathcal{T}), \{\bigwedge_{A \in \mathcal{T}} \exists y A(y)\}, \mathcal{T}, \varnothing)$.

Given a collection of MIPS $M = \{mips_1, \ldots, mips_n\}$, where each MIPS $mips_i$ is a set of axioms $ax_{1i}, \ldots, ax_{k_i i}$. The diagnoses for the incoherence problem is then the set of prime implicants of the Boolean formula:

$$\bigwedge_{1 \leq i \leq n} \bigvee_{1 \leq j \leq k} ax_{ij}$$

Figure 3.8 shows the hitting tree for the incoherence problem

$$(Sd(\mathcal{T}^*), \{\bigwedge_{A \in \mathcal{T}} \exists y A(y)\}, \mathcal{T}^*, \varnothing)$$

for $\mathcal{T}^*$ where minimal conflict sets have been calculated as MIPS.[3] This time there were only 12 nodes created.

[3] An axiom ax_i is represented by the number i.

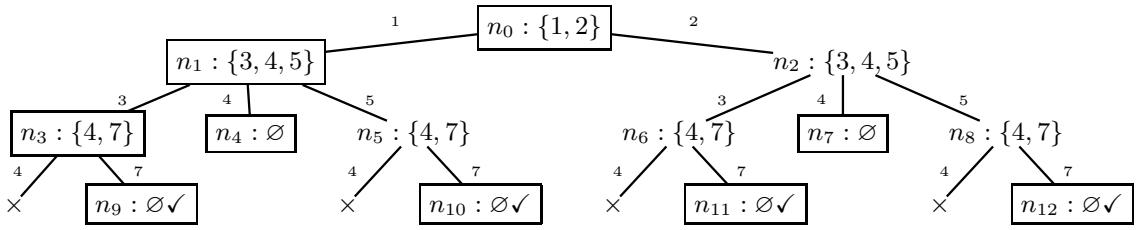

Figure 3.8: HS-Tree with minimal conflict sets

3.4 Pinpoints: approximating diagnosis

MIPSs, MUPS, MISOs and their cores do not offer an immediate recipe for fixing an incoherent web ontology. Diagnoses on the other hand can be very expensive to calculate (see the discussion on page 18). For practical reasons we therefore introduced the notion of a *pinpoint*, which is a small, but not necessarily minimal, set of axioms which have to be fixed or removed to make an ontology logically correct.

Here, we describe the idea to calculate pinpoints for MIPSs. The strategy iteratively calculates the cores (of size 1) of maximal arity, repeating the process on the remaining MIPSs. Moreover, the algorithms for MIPS and MUPS have been implemented for terminological debugging of $\mathcal{ALC}$ terminologies. To apply them to web ontologies we need some preprocessing. Given a web ontology $\mathcal{O}$ we apply the following steps:

1. Remove the ABox, as well as property statements and axioms where the left-hand side is non-atomic.
2. Replace equivalence statements by implications
3. Collect all implications $C \sqsubseteq D_1$,..., $C \sqsubseteq D_n$ in a single conjunction $C \sqsubseteq D_1 \sqcap \cdots \sqcap D_n$.
4. Call the resulting terminology $\mathcal{T}$ (not necessarily unfoldable)

The strategy for automatically fixing the incoherences by pinpointing is then as follows; calculate:

1. the set $Unsat(\mathcal{T})$ of unsatisfiable concept-names in $\mathcal{T}$ using RACER;
2. the MUPS $mups(\mathcal{T}, CN)$ for all concept-names $CN \in Unsat(\mathcal{T})$;
3. the MIPSs $mips(\mathcal{T})$ for $\mathcal{T}$ from the MUPS;
4. let $M := mips(\mathcal{T})$, $P(\mathcal{O}) = \varnothing$. Now calculate while $M \neq \varnothing$:
   (a) the core $\{ax\}$ of M of size 1 with maximal arity, and add it to $P(\mathcal{O})$;

(b) remove from M the MIPS containing ax.

5. $P(\mathcal{O})$ will be called the *pinpoint of* $\mathcal{O}$.

6. Finally, remove $P(\mathcal{O})$ from $\mathcal{T}$.

The pinpoint of the ontology $\mathcal{O}$ is a set of axioms in the preprocessed version. Every axiom corresponds precisely to a concept-name (because of step 3) or is a disjointness statement. By the pinpoint of an ontology, we will therefore refer both to sets of axioms as well as to sets of concept and disjointness statements. Note that for debugging and semantic clarification, we often focus on these pinpoints. This is because there is usually only a handful of those as compared to hundreds of MIPSs, which can be quite complicated. However, in practice, MIPSs will always have to be consulted if one wants to understand the underlying reasons for incoherence of an ontology.

Fixing an ontology by pinpointing offers a simple solution to the incoherence problem since by adjusting or removing the information about the pinpoints we can guarantee to the restoration of logical coherence with (almost) minimal intrusion in the ontology. Pinpoints correspond to hitting-sets [25] for the MIPS, but they are not necessarily minimal. Take, as example, a set $M = \{\{ax_1, ax_2\}, \{ax_3, ax_4\}, \{ax_5, ax_1\}, \{ax_5, ax_3\}\}$ of MIPS, with $\{ax_5, ax_1, ax_3\}$ as possible pinpoint, even though $\{ax_1, ax_3\}$ is a miminal covering set for M. On the other hand, it has to be remarked that calculating minimal hitting-sets from the set of MIPS is NP-COMPLETE, as compared to linear time to calculate pinpoints from the MIPS.

Chapter 4

Debugger of Inconsistent Ontologies: Prototypes

4.1 MUPSter: A Prototype for Top-Down Debugging

4.1.1 Implementation of MUPSter

We have implemented the MUPSter-system as a prototype for the top-down debugging of incoherent terminologies. The system implements the tableau-based approach described in section 3.1.1 to calculate the MUPS of an unsatisfiable concept, the three methods to calculate diagnoses for the incoherence problem, as well as cores, pinpoints and other debugging facilities.

MUPSter is implemented in JAVA, and is based on the WELLINGTON reasoner[1], making external calls to RACER for efficient "black-box" computations of unsatisfiable concepts. A given terminology is parsed and analysed, and iteratively debugged in the following order:

- **Unsatisfiable concepts:** a request to RACER is send to return a list of unsatisfiable concepts
- **MUPS:** for each unsatisfiable concept, the set of MUPS is calculated, using a specialised labelled tableau.
  - a minimisation function is calculated from the fully expanded tableau
  - the minimisation function is minimised with an implementation of a hitting-set algorithm to calculate prime-implicant.
  - these prime implicants are the MUPS.

[1] Wellington was developed at King's College, London [10]. It supports the standard `krss` format and the DIG language.

- **MIPS:** MIPS can easily be calculated from the MUPS of all unsatisfiable concepts.
- **Pinpoints and Cores** (as described in the previous section)
- **Diagnoses:** based on the MIPS (as minimal conflict sets) we can calculate diagnoses.

4.1.2 Installation and Test Guide

MUPSteris available as a JAVA jar archive, and can, in principal, be executed platform independent on machines running JAVA 1.5.[2]. MUPSter performs debugging on incoherent $\mathcal{ALC}$-terminologies. If the terminology is unfoldable, the algorithms calculate the set of all MUPS, MIPS and diagnoses. Otherwise, the algorithm is incomplete, and the results can often be not very useful.

MUPSterstill is an experimental prototype, and in some cases problems will most probably occur. However, installing and running MUPSter is very easy, simply follow the instructions below:

- **Download** MUPSter**:** Download the file `mupster.jar` from

  `\url{http://www.wasp.cs.vu.nl/sekt/mupster}`

- **Pre-process ontology:** MUPSter's parsers are for the time being not very robust, and the language accepted is rather restricted. On the previously mentioned website, some scripts are available for convenience to transform a krss ontology into format where problems could be avoided.
- **Download RACER:** MUPSter requires RACER to run as a server as external DL reasoner. RACER can be downloaded from the website: `http://www.sts.tu-harburg.de/~r.f.moeller/racer/download.html`
- **Start RACER:** The RACER server has to be started and listen to the default port: 8000
- **Run** MUPSter**:** Under Linux run MUPSter with the following command line:

  `java -jar mupster.jar -f file.krss (options)`
  for ontologies in krss-format
  `java -jar mupster.jar -f file.dig -dig (options)`
  for ontologies in dig-format

  where `options = [ -u unsatfile ] [-v] [-r] [-o]` . It is recommended not to use the -r and -o options.

[2]The system has been developed and tested under Linux, and there are known problems to run MUPSter under Windows.

Description

MUPSter parses an ontology according to the file specification (krss is the default). It then starts debugging and diagnosis depending on the options that are set. In a file `unsatfile` the user can specify the unsatisfiable concept-names as a comma-free list.[3] The other options are

-v MUPSter runs in `verbose` mode, and returns more (intermediate) debugging information.

-r MUPSter *only* calculates *diagnosis* with Reiter's algorithm with **maximal conflict sets**. Only use this option on small, but expressive ontologies. Note, that the ontology has to be extremely small, be small because the method is extremely slow.

-r -o MUPSter *only* calculates *diagnosis* with Reiter's algorithm with **small conflict sets**. The method is still quite slow, and only works for $\mathcal{ALC}$-terminologies.

In all other cases, MUPSter currently calculates the full set of debugging operations: MUPS, MIPS, pinpoints and diagnosis.

Output

MUPSter prints a debugging analysis of the original terminology to `<STOUT>`. The output of MUPSter on our example terminology $\mathcal{T}^*$ from section 2 is as follows:

```
> there are 12 concept names of which 4 are incoherent

0th Mups(A6) = [(and A3 A1 A4 A5 A6), (and A6 A1 A2 A4)]
1th Mups(A1) = [(and A1 A2), (and A3 A1 A4 A5)]
2th Mups(A3) = [(and A3 A4 A5)]
3th Mups(A7) = [(and A7 A4)]

> Mips(T): (and A1 A2)
           (and A3 A4 A5)
           (and A7 A4)

Core: {A4} of arity 2 with definition (and (all S B) C)
Core: {A1} of arity 1 with definition (and (not A) A2 A3)
Pinpoint: [A4, A1]
```

[3] Note that the names of the concepts have to be the same as those MUPSter **returns**. In order to avoid any problems with this option run MUPSterwith the verbose option [-v], and copy the list of unsatisfiable concepts into a single file `unsatfile`. From this file you will have to remove the commas.

```
Calculating Diagnoses from MIPS
> Diagnoses(T): [A1, A4]
                [A2, A4]
           [A1, A3, A7]
           [A1, A5, A7]
           [A2, A3, A7]
           [A2, A5, A7]

Time for Debugging: 567ms
```

4.2 DION: a prototype for bottom-up debugging

4.2.1 Implementation of DION

We have implemented a prototype DION as a debugger of inconsistent ontologies. The system is implemented as an intelligent interface between an application and state-of-the art description logic reasoners and provides server-side functionality in terms of an XML-based interface for uploading an inconsistent ontology and posing queries for debugging. Requests to the server are analysed by the main control component that also transforms queries into the underlying query processing. The main control element also interacts with the ontology repository and ensures that the reasoning components are provided with the necessary information and coordinates the information flow between the reasoning components.

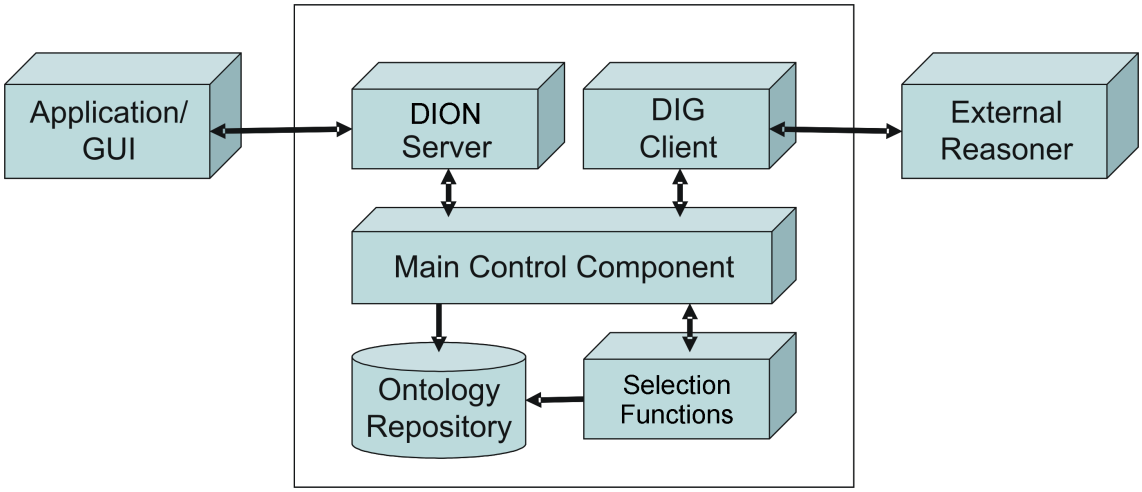

Figure 4.1: Architecture of DION.

An overview of the DION architecture is shown in Figure 4.1. It has the following components:

- **DION Server**: The DION server acts as a server which deals with requests from other ontology applications.
- **Main Control Component**: The main control component performs the main processing, like query analysis, query pre-processing, and interacting with the ontology repositories.
- **Selection Functions**: The relevance-based selection function component provides heuristic facilities to evaluate the queries.
- **DIG Client**: DION's DIG client is the standard XDIG client, which calls external DL reasoners which support the DIG interface to obtain the standard DL reasoning capabilities.
- **Ontology Repositories**: The ontology repositories are used to store ontologies and other system specifications.

The DION prototype is implemented in Prolog and uses the XDIG interface [17], an extended DIG description logic interface for Prolog[4]. DION is designed to be a simple API for a general debugger with inconsistent ontologies. It supports extended DIG requests from other ontology applications, including OWL and DIG[4][5]. This means that DION can be used as an interface of an inconsistent ontology debugger as it supports the functionality of the underlying reasoner by just passing requests on and provides reasoning functionalities if needed. Therefore, the implementation of DION will be independent of those particular applications or systems.

4.2.2 Functionalities

The current version of the DION prototype has the following characteristics:

- **Debugging Support**: calculates MUPSs, MIPSs, cores, and pinpoints
- **Interface Support**: supports the XDIG interface, an extended DIG interface.
- **Ontology Languages**: supports the DIG format as well as the OWL language. Ontological data in the OWL format is translated automatically by the XDIG component 'owl2dig'[6].

[4] `http://wasp.cs.vu.nl/sekt/dig`

[5] `http://dl.kr.org/dig/`

[6] However, note that the component 'owl2dig' is still under development. The complete specification of OWL DL is not yet supported .

4.2.3 Installation and Test Guide

1. **Download**: The DION package is available from the DION website: `http://wasp.cs.vu.nl/sekt/dion/` Unzip the DION package into a directory. We will call the directory $DION\_ROOT$.

2. **Installation of SWI-Prolog**: DION requires that SWI-Prolog (version 5.4.7 or higher) has been installed on your computers. It can be downloaded from the SWI-Prolog website: `http://www.swi-prolog.org` Install SWI-Prolog into a directory. We will call that directory $SWIPROLOG\_ROOT$.

3. **Installation of XDIG**: You can find the zip file 'diglibrary.zip', the XDIG libraries in the directory $DION\_ROOT$. Unzip the file 'diglibrary.zip' into the SWI-Prolog library directory, i.e., $SWIPROLOG\_ROOT/library$.

4. **Installation of RACER**: DION requires RACER (version 1.7.14 or higher) as its external DL reasoner. Other DL reasoners may work for DION if they support the DIG DL interface, however, they have not yet been tested. RACER can be downloaded from the website: `http://www.sts.tu-harburg.de/~r.f.moeller/racer/download.html`

The DION testbed 'dion_test.htm' is a DION client with a graphical interface, which is designed as a webpage. Therefore it can be launched from a web browser which supports Javascript. A screenshot of the DION testbed, is shown in Figure 4.2.

The current version of the DION testbed supports tests for which both the DION server (with default port: 8004) and the external DL reasoner (with the default port: 8000) are running on the localhost. The default hostname of the DION server and the external DL reasoner is 'localhost'. For a DION server which runs on a remote host, change the host and port data in the 'dionmain.htm' file. Namely, replace the URL 'http://127.0.0.1:8004' with another valid URL.

Before starting the DION test, make sure that the DION server and the external DL reasoner (i.e. RACER) are running at the host with the correct ports.

1. **Launch RACER**: Racer can be launched by the following command: racer -http 8000 Alternatively, click on the file 'racer8000.bat' if the DION download package includes the reasoner RACER and the batch file.

2. **Launch DION server**: click on the file '$dion\_server.pl$' in the DION directory. If you encounter the global stack limit problem because of a big amount of test data, you should increase the size of the global stack. The windows users can edit the path setting of 'plwin.exe' in the file 'dionserver_bigGlobalStack.bat', then launch it.

Figure 4.2: DION testbed

The TELL and ASK request data can be copied into the TELL text area and the ASK text area respectively. After that, you can click on the buttons 'Tell' and 'Ask' to make the corresponding requests. The request data can also be posted from any application without using the DION testbed. That is useful if the test data exceeds the text area limit.

The MadCow Ontology[7] is an inconsistent ontology in which MadCow is stated as a Cow which eats brains of Sheep, whereas Cow is considered as a Vegetarian. DION finds the four pinpoints: (i) either removing the statement 'Cows are vegetarians', or (ii) removing the statement which defines MadCow as a Cow which eats brains of sheep, or (iii) removing the statement 'sheep are animals', or (iv) removing the statement which defines Vegetarian as one which never eats parts of animals. The DION pinpoints of the MadCow ontology are shown in Figure 4.3.

[7] `http://www.daml.org/ontologies/399`

Figure 4.3: Pinpoints of the MadCow Ontology

Chapter 5

Related Work

Dealing with inconsistency in applications and logics has a long tradition, and the field of model-based diagnosis, for example, looks back at over 20 years of history. In contrast their is relatively little work on logical modelling support of ontologies, and work on explanation of reasoning has only become popular in the last few years.

In [14], a framework of reasoning with inconsistent ontologies, in which pre-defined selection functions are used to deal with concept relevance, is presented. The notion of ”concept relevance” is used for reasoning with inconsistent ontologies. A prototype called PION (Processing Inconsistent ONtologies), which is based on a syntactic relevance-based selection function is implemented. In this document, we have used the concept relevance, a technique which is developed in PION, for the informed bottom-up approach of inconsistent ontology diagnosis. In [12], Haase et al. survey four different approaches to handling inconsistency in DL-based ontologies: consistent ontology evolution, repairing inconsistencies, reasoning in the presence of inconsistencies and multi-version reasoning. They present a common formal basis for all of them, and use this common basis to compare these approaches. They discuss the different requirements for each of these methods, the conditions under which each of them is applicable, the knowledge requirements of the various methods, and the different usage scenarios to which they would apply.

In [20], Meyer et al. propose to recast techniques for propositional inconsistency management into the description logic setting. They show that the additional structure afforded by description logic statements can be used to refine these techniques. Their approach focus on the formal semantics for such techniques, although they do provide high-level decision procedures for the knowledge integration strategies discussed.

Recently, the Description Logic community has started to work on explanation of reasoning. From a more practical point of view, closest to our work are the Chimaera and PROMPT tools described in [18] and [22], which provide support for the merging, analysis and diagnosis of a knowledge base but not, to our knowledge, for debugging. Despite a significant interest in explanation of DL reasoning recently shown in the DL community (as [9] suggests) relatively little work has been published on the subject. One excep-

tion is [19] where the author provides explanation for subsumption and non-subsumption. Her approach, based on *explanation as proof fragments*, uses structural subsumption for CLASSIC and has been extended to $\mathcal{ALC}$-tableau reasoning in [5]. In contrast to this approach, our non-standard reasoning services for axiom pinpointing focus on the reduction of information, and are independent of particular calculi or implementations. In [23], Parsia et al. have integrated a number of debugging cues generated from their reasoner, Pellet, in their hypertextual ontology development environment, Swoop. These debugging cues, in conjunction with extensive undo/redo and Annotea based collaboration support in Swoop, significantly improve the OWL debugging experience, and point the way to more general improvements in the presentation of an ontology to users. However, their explanation facilities are very simple, as they only point to the "last" contradiction in an unsatisfiability proof, and do not have a pinpointing facility in case of more complicated or multiple contradictions.

In [3], Baclawski et al. introduce the ConsVISor tool for consistency checking of ontologies.This tool is a consistency checker for formal ontologies, including both traditional data modelling languages and the more recent ontology languages. ConsVISor checks consistency by verifying axioms. ConsVISor is part of the UBOT toolkit that uses a variety of techniques such as theorem proving and logic programming.Some examples of the use of these tools are given in [3].

Finally, we should mention that the techniques, that we use to calculate MIPSs, however, are similar to those introduced in [1], where the authors use Boolean minimisation to calculate minimal inconsistent ABoxes to construct an implicit minimal model for defaults.

Chapter 6

Discussion and Conclusion

Summary

With the arrival of more expressive ontology languages in the Semantic Web community, logical modelling errors increasingly become a problem that can seriously hamper the construction and application of ontologies. As a first step of logical modelling support in such cases we introduced a framework for *pinpointing*, which is based on the idea that modelling errors can best be detected in ontologies of minimal size still containing the contradiction. Secondly, the principle of parsimony suggests that an ideal fix of an erroneous ontology is to consider *diagnoses*, i.e. minimal sets of axioms which are to be repaired to render the ontology logically correct.

Pinpointing The first element of our framework to deal with incoherence and inconsistency in ontologies was to present a strategy for automatically identifying and fixing incoherences that is based on the explaining of their causes and, secondly, choosing (and eliminating) axioms that most frequently participate in the underlying logical contradictions. We have introduced the non-standard reasoning services MIPS, MUPS, MISOs and pinpoints, which facilitate the identification of error-relevant minimal subsets of ontological axioms in DL terminologies and ontologies, and the subsequent debugging of logically inconsistencies.

As we investigated debugging as an integral part of the practical development of a realistic terminology, we developed effective algorithms to calculate MIPS, MUPS and pinpoints. To determine MUPS, there are two types of algorithms, both were implemented prototypically. The first type, a top-down analysis of a closed labelled tableau with Boolean minimisation, was developed for the particular application of the DICE terminology. Therefore, some restrictions apply, most importantly the restriction to unfoldable $\mathcal{ALC}$ TBoxes. Neither is essential, and we conjecture that it is not too hard to extend our algorithms to find MIPS both for more expressive languages and for general TBoxes. Both issues will be addressed in future investigations. The other method for

calculating MUPS is bottom-up: we enumerate terminologies by size, and check for incoherence until we find a minimal one. The advantage of this second method is that it is universally usable for ontologies in any language for which a generic consistency checker exists, e.g. $\mathcal{SHIQ}(D)$ for the RACER reasoner, and that there are no restrictions on the type of axioms. The disadvantage is that the algorithms are not complete, i.e. not all MUPS are guaranteed to be found. Furthermore, we expect a specialised algorithm to perform better, a claim which we will further investigate in the upcoming SEKT Deliverable, 3.6.2.

Diagnosis The second, orthogonal, approach to dealing with incoherence and inconsistency of ontologies is based on model-based diagnosis. Representing the terminological incoherence problems as a first-order system descriptions allows us, at least in theory, to use Reiter's general framework to calculate diagnoses for very expressive terminologies. There are several conceivable extensions: one can use diagnoses to explain correct and incorrect subsumption. Given a coherent terminology, one can specify subsumption (or non-subsumption) as expectations. Similarly, extending diagnosis with instances is easy, one simply has to add assertional axioms to the system description, and instance relations to the expectations. In all cases, diagnosis works "off-the-shelf" as long as the components are set of axioms. But even this leaves room for extension, as one can easily choose particular subsets of a terminology as sets of components, which not only can be very useful in practice, but can improve the efficiency significantly.

This will be necessary, as our experimental evaluation shows that the complexity of the general problem is so high that it is doubtful whether it will work on large incoherent terminologies. Only the two more specialised algorithms work in practice, which implies that implicit information on proofs is required. We believe that it should be feasible to extract small conflict sets from a more verbatim output of current DL reasoner. Applying an optimised, efficient and expressive general-purpose reasoning system for small conflict set creation has many advantages, as it would make the much more efficient method based on small conflict sets available for terminologies such as WINE or MadC. Given our bottom-up method to create MUPS, we plan to combine the advantages of the three approaches described in this paper; taking generic algorithm for debugging of incoherent terminologies, which should be reasonably efficient, and applicable on expressive ontologies.

Relation with SEKT 3.4.1: Reasoning with inconsistent ontologies

In an earlier SEKT Deliverable (D3.4.1, [16]), we presented an entirely different approach to dealing with inconsistent ontologies. There, we presented a framework of reasoning with inconsistent ontologies, in which pre-defined selection functions are used to select subsections of the ontology which are both sufficiently large to answer a given query, and sufficiently small to avoid containing any inconsistencies. Although there are technical

similarities between that work and the work presented here (the use of the notion of relevance in the bottom-up approach of section 3.2), the two approaches are conceptually different, and in fact complementary: D3.4.1 tries to "live" with the inconsistencies, and do query-answering as good as possible in the presence of them, while D3.4.2 tries to remove the inconsistencies (through a process of diagnosis and repair), so that queries can be answered on the repaired, consistent ontology. It is natural to ask when one should choose for which approach. Although more practical experience is required in this matter, we would argue that this decision depends mainly on the use-case at hand.

For example, in a typical Semantic Web setting, one would be importing ontologies from other sources, making it impossible to repair them, and the scale of the combined ontologies may be too large to make repair effective.

Also, it may not always be possible for users of an ontology to decide on the correct repair-actions for an inconsistency, or even to choose between the alternative diagnoses for an inconsistency.

On the other hand, for an ontology owner, it might make sense to debug on ontology "off-line", before intensive use by others (the DICE scenario mentioned in chapter 1 is an example of this use).

A further consideration is the fact that the "reasoning with inconsistencies" approach of D3.4.1 is inherently heuristic: no guarantee can be given whether it is possible for a given query to determine a consistent subset of the ontology large enough to answer the query. The approach of section 3.2 on the other hand are algorithmic, and are guaranteed to find the inconsistencies if they exist.

This discussion is by no means exhaustive, but serves to indicate that the choice between the complementary approaches of the earlier D3.4.1 and the current D3.4.1 is indeed use-case dependent.

Future Work

The research described in this paper raise a number of challenging technical issues, mostly related to performance of the algorithms: first we will have to address the computational problems by improving the HS-Tree implementation as well as the tableau calculus of our DL engine for the top-down method. One possible line of attack on this is to investigate the integration of our algorithms in existing high-performance DL-reasoners such as Fact++ or Pellet. Along similar lines, we could consider to make MUPSter and DION available as a Protégé. This would make the SEKT technology available to a much wider audience. As another important step in this direction we will in SEKT Deliverable 3.6.2 perform an extensive evaluation of the tools we presented here.

Preliminary experiments and experience on the debugging on incoherent terminologies in our practical applications, e.g. on the DICE terminology, have shown promising results. Although we failed in some cases to calculate diagnoses, we almost always re-

trieved MIPS in reasonable time (within minutes even for complex terminologies with several thousand concepts and hundreds of unsatisfiable concepts). More problematic was that the choice of axioms as components, which was perceived as too coarse in practise. More precisely, the notion of a MIPS as a subset of a terminology is often not sufficient to explain the logical contradiction, because of their complexity, and the users of the system, in our case domain experts in medical modelling expressed an urgent interest in more fine-grained debugging. Some first ideas for this have been presented in [6]. An alternative research direction is to extend the functionality of the tool, for example by explaining subsumption and instance relations with the methods described above.

On a higher level, there are many interesting relations of diagnosis and debugging with other problems, that we plan to address, including to integrate the diagnosis approach with reasoning with inconsistent and approximated ontological reasoning. The potential is easily seen, as we determine minimal sub-ontologies (e.g. when calculating diagnoses) which we can be use for reasoning instead of the original, inconsistent, ontology. This immediately suggests a first way of reasoning with inconsistency, and gives rise to new approximation techniques.

From an orthogonal perspective, new challenges to debugging and diagnosis arise when we move from single ontologies to distributed or multi-version settings. Here, logical incorrectness can occur in mappings or when ontologies are merged, so that questions of preference of axioms (for example between local and global ontologies) or between different versions of ontologies have to be taken into account.

A further, and again orthogonal set of challenges arises from possible variations in the language used to represent the ontologies. In this report, we have concentrated on OWL DL as the language in which to express the ontologies. However, with the advent of rule-languages on the Semantic Web, the question arises if our techniques can be extended to include such more expressive languages. As explained in section 3.4, the calculation of explanations varies between languages, depending on the expressiveness of specialised algorithms (for our top-down methods) or of the general purpose reasoning tools such as RACER (for our bottom-up approach). In principle, the general bottom-up approach can be applied to any language for which there is a sufficiently efficient reasoner available. This suggests that it will not be possible *in general* to diagnose rule-sets in the same way as we have diagnoses OWL DL ontologies, since many of the current candidates for Semantic Web rule-languages are no longer even decidable, and hence it would not even be possible *in general* to notice there inconsistency. This is simply an instance of the general trade-off between what one can express in a language on the one hand (its expressivity), and the computations that such a language allows on the other hand. The undiagnosability *in general* of rule-sets will have to be taken into account. Nevertheless, it may well be the case that many rule-sets that will be written *in practice* turn out to be diagnosable. This is a matter of further empirical research, that can only be settled when such rule-sets will become available.

A second, related issue is the extension of our methods from terminological reasoning

only to full ontology debugging. Chapter 3 presented two general approaches to calculate explanations: a top-down method, which reduces the reasoning into smaller parts in order to explain a subproblem with reduced complexity, and an informed bottom-up approach, which enumerates possible solutions in a clever way. Both can, in principal, easily be extended to full ontology debugging. The extension of the bottom-up method is trivial, as we only have to define a new selection function and systematic enumeration. For the top-down approach things are a bit more complicated, as we now have analyse forests rather than trees. However, this seems to be a technical rather than a conceptual problem.

In this work, we have diagnosed ontologies that were expressed OWL DL.

Bibliography

[1] F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. Technical Report RR-93-20, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1993.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Kenneth Baclawski, Mieczyslaw M. Kokar, Richard Waldinger, and Paul A. Kogut. Consistency checking of semantic web ontologies. In *Proceedings of ISWC2002*, 2002.

[4] Sean Bechhofer, Ralf Möller, and Peter Crowther. The dig description logic interface. In *International Workshop on Description Logics (DL2003)*. Rome, September 2003.

[5] A. Borgida, E. Franconi, and I. Horrocks. Explaining $\mathcal{ALC}$ subsumption. In *Proc. of the 14th Eur. Conf. on Artificial Intelligence*, pages 209–213, 2000.

[6] R. Cornet, S. Schlobach, and A. Abu-Hanna. Knowledge representation with ontologies: Present challenges - future possiblities. *submitted to the International Journal Of Human Computer Studies*, 2005.

[7] Ronald Cornet and Ameen Abu-Hanna. Evaluation of a frame-based ontology. A formalization-oriented approach. In *Proceedings of MIE2002.*, volume 90, pages 488–93, 2002.

[8] Maria Garcia de la Banda, Peter J. Stuckey, and Jeremy Wazny. Finding all minimal unsatisfiable subsets. In *Fifth ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 32–43. ACM, 2003.

[9] Minutes of the DL Implementation Group Workshop, 2002. `http://dl.kr.org/dig/minutes-012002.html`, visited on January 9, 2003.

[10] U. Endriss. Reasoning in description logics with WELLINGTON 1. 0. In *Proceedings of the Automated Reasoning Workshop 2000*, 2000.

[11] Volker Haarslev and Ralf Möller. Description of the racer system and its applications. In *Proceedings of the International Workshop on Description Logics (DL-2001)*, pages 132–141. Stanford, USA, August 2001.

[12] Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. A framework for handling inconsistency in changing ontologies. In *Proceedings of ISWC2005*, 2005.

[13] I. Horrocks. The FaCT system. In *TABLEAUX 98*, pages 307–312, 1998.

[14] Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI'05*, 2005.

[15] Z. Huang, F. van Harmelen, A. ten Teije, P. Groot, and C. Visser. Reasoning with inconsistent ontologies: a general framework. Project Report D3.4.1, SEKT, 2004.

[16] Zhisheng Huang and Frank van Harmelen. Reasoning with inconsistent ontologies: a general framework. Deliverable D3.4.1, SEKT, 2004.

[17] Zhisheng Huang and Cees Visser. Extended dig description logic interface support for prolog. Deliverable D3.4.1.2, SEKT, 2004.

[18] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. The chimaera ontology environment. In *The Seventeenth National Conference on Artificial Intelligence*, 2000.

[19] Deborah McGuinness. *Explaining Reasoning in Description Logics*. PhD thesis, Department of Computer Science, Rutgers University, 1996.

[20] Thomas Meyer, Kevin Lee, and Richard Booth. Knowledge integration for description logics. In *Proceedings of The 7th International Symposium on Logical Formalizations of Commonsense Reasoning*, 2005.

[21] B. Nebel. Terminological reasoning is inherently intractable. *AI*, 43:235–249, 1990.

[22] N. Now and M. Musen. PROMPT: Algorith and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*. AAAI Press, 2000.

[23] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging owl ontologies. In *The 14th International World Wide Web Conference (WWW2005)*, 2005.

[24] W.V. Quine. The problem of simplifying truth functions. *American Math. Monthly*, 59:521–531, 1952.

[25] R. Reiter. A theory of diagnosis from first principles. *Artif. Intelligence*, 32(1):57–95, 1987.

[26] S. Schlobach. Semantic clarification by pinpointing. In *Proceedings of ESWC2004*, 2004.

[27] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the eighteenth International Joint Conference on Artificial Intelligence, IJCAI'03*. Morgan Kaufmann, 2003.