D1.2.2.1.4 Benchmarking of Processing Inconsistent Ontologies

Zhisheng Huang (Vrije Universiteit Amsterdam)

with contributions from:
Johanna Volker, Qiu Ji(University of Karlsruhe),
Heiner Stuckenschmidt, Christian Meilicke(University of Mannheim),
Stefan Schlobach, Frank van Harmelen(Vrije University Amsterdam),
and Joey Lam(University of Aberdeen)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D2.1.6.3/D1.2.2.1.4 (WP2.1+WP1.2)
This deliverable investigates methods and results for benchmarking of processing inconsistent ontologies. In this document, we propose a gold standard specification language for evaluation of processing inconsistent ontologies. We have implemented a benchmarking suite for processing inconsistent ontologies, which consists of benchmarking tools, data sets, and gold standards. We have performed a series of experiments of benchmarking with realistic inconsistent ontologies. In this document, we report a comprehensive evaluation of various approaches of processsing inconsistent ontologies.
Keyword list: benchmarking, ontology reasoning, inconsistency

Document Identifier	KWEB/2007/D1.2.2.1.4/v1.0.0
Project	KWEB EU-IST-2004-507482
Version	v1.0.0
Date	January 3^{rd}, 2008
State	final draft
Distribution	public

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

University of Innsbruck (UIBK) - Coordinator
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

École Polytechnique Fédérale de Lausanne (EPFL)
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

France Telecom (FT)
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Freie Universität Berlin (FU Berlin)
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

National University of Ireland Galway (NUIG)
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

The Open University (OU)
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Karlsruhe (UKARL)
Institut für Angewandte Informatik und Formale
Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Liverpool (UniLiv)
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Manchester (UoM)
Room 2.32. Kilburn Building, Department of Computer
Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

University of Trento (UniTn)
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Amsterdam (VUA)
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

Vrije Universiteit Brussel (VUB)
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

University of Karlsruhe
University of Aberdeen
Vrije Universiteit Amsterdam
University of Mannheim

Executive Summary

Re-using and combining multiple ontologies on the Web is bound to lead to inconsistencies between the combined vocabularies. Even many of the ontologies that are in use today turn out to be inconsistent once some of their implicit knowledge is made explicit. That appeals for efficient and robust approaches to deal with inconsistencies in the Semantic Web.

Various frameworks of processsing inconsistent ontologies have been proposed, which range from various methods for reasoning with inconsistent ontologies to various approaches for debugging inconsistent ontologies. In order to enable a user or a system developer to decide which method is best suited for his/her task, we need a comprehensive evaluation and benchmarking on those proposed approaches.

This deliverable investigates methods and results for benchmarking of processing inconsistent ontologies. First we present a methodology study of the benchmarking of processing inconsistent ontology. We develop a gold standard specification language for automatic/semi-automatic evaluation of processing inconsistent ontologies. We have implemented a benchmarking suite for processing inconsistent ontologies. In this document we provide a detailed manual how to use the benchmarking suite.

We have performed a series of experiments of benchmarking with realistic and large scale inconsistent ontologies. In this document, we report a comprehensive evaluation of various methods of processing inconsistent ontologies, which include a) syntactic approaches versus semantic approaches, b) linear extension versus multi-step extension, c) blind backtracking versus informed backtracking, and d) reasoning with inconsistent ontologies versus debugging inconsistent ontologies. Those methods are evaluated with respect to the three benchmarking factors: quality of query answers, performance, and scalability. We finally discuss the results and draw the conclusions about the future of processing inconsistent ontologies.

Contents

1 Introduction **1**

2 Processing of Inconsistent Ontologies **5**
2.1 General Framework of Reasoning with Inconsistent Ontologies 5
2.2 Strategies 6
2.3 Relevance-based Selection Functions 8
2.4 Syntactic Relevance-based Selection Functions 9
2.5 K-extension 10
2.6 Semantic Relevance Based Selection Functions 12
2.7 Variants of Over-determined Processing 17
2.7.1 Pros and Cons of Semantic Relevance 18
2.7.2 Mixed Approach for Over-determined Processing 18
2.8 Debugging Incoherent Terminologies 19
2.8.1 Logical errors in Description Logic terminologies 20
2.8.2 Framework for debugging and diagnosis 21
2.8.3 DION: A Bottom-up Approach for Debugging Incoherent Ontologies 25
2.8.4 RepairTab: A Heuristic Approach for Repairing Unsatisiable Ontologies 28
2.8.5 RaDON: A System for Reasoning and Diagnosis in Ontology Networks 30
2.8.6 Other Approaches for Debugging 30

3 A Framework for Benchmarking of Processing Inconsistent Ontologies **32**
3.1 Framework 32
3.1.1 Measuring the Quality of Query Answers 32
3.1.2 Basic Definitions 33
3.1.3 Workflows of Evaluation and Benchmarking 34
3.1.4 Taxonomy 34
3.2 A Specification Language for Golden Standards 36
3.3 A Benchmarking Suite for Processing Inconsistent Ontologies 38
3.3.1 Benchmarking Tools 38
3.3.2 Gold Standard Authoring Tool 39

3.3.3 Test Tool 40
3.3.4 Evaluation Tool 42
3.4 Data Sets 45
3.4.1 Inconsistent Mapping 46
3.4.2 Inconsistency created by Ontology Learning 48

4 Benchmarking Experiments **51**
4.1 Syntactic Approaches versus Semantic Approaches 51
4.2 Linear extension versus Multi-step extension 55
4.3 Reasoning with Inconsistent Ontologies versus Debugging of Inconsistent Ontologies 58
4.4 Inconsistency Processing and Reasoners 61

5 Discussions and Conclusions **63**
5.1 Discussions 63
5.1.1 Semantic Approach for Reasoning with Inconsistent Ontologies . 63
5.1.2 Integrating Reasoning with Inconsistent Ontologies with Debugging of Inconsistent Ontologies 64
5.1.3 Processing Inconsistencies in Database Systems 64
5.2 Future Work 65
5.3 Concluding Remarks 66

Chapter 1

Introduction

One cannot live without inconsistency.
- Carl Gustav Jung (1875-1961)

There is nothing constant in this world but inconsistency.
-Jonathan Swift (1667-1745)

A key ingredient of the Semantic Web vision is avoiding to impose a single ontology. Hence, merging ontologies is a key step. Earlier experiments (e.g. [Hameed *et al.*, 2003]) have shown that merging multiple ontologies can quickly lead to inconsistencies. Other studies have shown how migration [Schlobach and Cornet, 2003a] and evolution [Haase *et al.*, 2005] also lead to inconsistencies. This suggests the importance and omnipresence of inconsistencies in ontologies in a truly web-based world.

At first sight, it might seem that many ontologies are semantically so lightweight (e.g. expressible in RDF Schema only, [d'Aquin *et al.*, 2007]) that the inconsistency problem doesn't arise, since RDF Schema is too weak to even express an inconsistency[1]. However, [Schlobach, 2005a] has shown that on a closer look, many of these semantically lightweight ontologies make implicit assumptions such as the Unique Name Assumption, or assuming that sibling classes are disjoint. Such implicit assumptions, although not stated, are in fact used in the applications that deploying these ontologies. Not making these disjointness assumptions explicit harms the re-usability of these ontologies. However, if such assumptions are made explicit, many ontologies turn out to be in fact inconsistent.

In [Volker *et al.*, 2007a], Volker and her colleagues made an experiment of human annotation of disjointness. In this experiment, we observe an interesting case which shows that inconsistency occurs easily even in medium-scale ontologies with single authorship in a non-distributed environment.

In the experiment of the disjointness by Volker and her colleagues, a large number

[1] besides the rather limited case of disjoint datatypes

of manually created disjointness are manually created. As a basis for the creation of the datasets and as background knowledge for the ontology learning algorithms they took a subset of the freely available PROTON ontology. Each concept pair was randomly assigned to 6 different people - 3 from each of two groups the first one consisting of PhD students from their institute (all of them professional "ontologists"), the second is being composed of under-graduate students without profound knowledge in ontological engineering. Each of the annotators was given between 385 and 406 pairs along with natural language descriptions of the classes whenever those were available. Possible taggings for each pair were $+$ (disjoint), $-$ (not disjoint) and ? (unknown). Furthermore, they computed the majority votes for all the above mentioned datasets by considering the individual taggings for each pair. If at least 50 percent (or 100 percent respectively) of the human annotators agreed upon $+$ or $-$ this decision was assumed to be the majority vote for that particular pair. See [Volker *et al.*, 2007a] for the details of the experiments.

Adding those created disjointness to PROTON results in inconsistent PROTON ontologies. For example, there are 24 unsatisfiable concepts in the PROTON ontology with the disjointness are created by 50 percent votes by the students. It is more interesting to observe that 100 percent of experts and students agree at the following axioms, however, they are inconsistent[2]:

$\{Reservoir \sqsubseteq Lake,$
$Lake \sqsubseteq WaterRegion,$
$Reservior \sqsubseteq HydrographicStructure,$
$HydrographicStrure \sqsubseteq Facility,$
$Disjoint(WaterRegion, Facility)\}.$

The conclusion $Reservoir \sqsubseteq WaterRegion$ follows from the first two axioms, whereas $Reservoir \sqsubseteq Facility$ follows from the third and the fourth axioms. However, it contradicts with the axiom that $WaterRegion$ and $Facility$ are disjoint. This case shows that inconsistency occurs much more easily than what people expect, because this scenario involves with only a small size of ontology without multiple-authorship and distribution. It is surprising that the expertise does not help to avoid the inconsistency in the ontologies.

Considering the fact that most realistic ontologies in the Semantic Web involve scalability, distribution, and multi-authorship, inconsistencies are expected to occur much more frequently if those ontologies are required to provide suitable formalization of a conceptualization, like enriched ontologies with disjointness axioms or negation concepts. Therefore, dealing with inconsistencies is one of important issues in the Semantic Web.

One way to deal with inconsistencies is to first diagnose and then repair them. [Schlobach and Cornet, 2003a] proposes a nonstandard reasoning service for debugging inconsistent terminologies. This is a possible approach, if we are dealing with one ontology and we would like to improve this ontology. Another approach to deal with in-

[2]In this document we do not make a distinction between inconsistency and incoherence. Namely, they are interchangeable. See [Flouris *et al.*, 2006] for the distinction between inconsistency and incoherence.

consistent ontologies is to simply avoid the inconsistency and to apply a non-standard reasoning method to obtain answers that are still meaningful, even though they have been obtained from an inconsistent ontology [Huang *et al.*, 2005, Huang *et al.*, 2004]. The first approach could be dubbed "removing inconsistencies", while the second could be called "living with inconsistencies". This latter approach is more suitable for an open Web setting, where one would be importing ontologies from other sources, making it impossible to repair them, and where the scale of the combined ontologies would be too large to make repair effective. PION is a system of reasoning with inconsistent ontologies which is developed based the latter approach[Huang *et al.*, 2005, Huang *et al.*, 2004].

The classical entailment in logics is *explosive*: any formula is a logical consequence of a contradiction. Therefore, conclusions drawn from an inconsistent knowledge base by classical inference may be completely meaningless. The general task of a system of reasoning with inconsistent ontologies is: given an inconsistent ontology, return *meaningful* answers to queries. In [Huang *et al.*, 2005, Huang *et al.*, 2004] a general framework of reasoning with inconsistent ontologies is developed. Various approaches of relevance based selection functions, extension strategies, and various approaches of overdetermined processing are proposed for processing inconsitent ontologies to obtain meaningful answers, where the meaningfulness is interpreted as the answer is supported by a selected consistent sub-ontology of the inconsistent ontology, and its negative answer is not supported.

In [Huang and van Harmelen, 2006, Huang and van Harmelen, 2007, Schlobach *et al.*, 2006, Schlobach and Huang, 2007], the preliminary experiments of various approaches are evaluated and the results have been reported. In order to obtain better understanding of the performance and the anwer quality of various approaches of processing inconsistent ontologies, in this document, we report experiments and evaluation of various approaches with several realistic ontologies.

In order to justify the benchmarking approach, a methodology study of benchmarking is helpful. This document presents a methodology study of the benchmarking of processing inconsistent ontology. We develop a gold standard specification language for automatic/semi-auotmatic evaluation of processing inconsistent ontologies. We have implemented a benchmarking suite for processing inconsistent ontologies. In this document we will provide a detailed manual for how the benchmarking suite can be used.

We have performed a series of experiments of benchmarking with realistic and large scale inconsistent ontologies. In this document, we report a comprehensive evaluation of various methods of processing inconsistent ontologies, which include a) syntactic approaches versus semantic approaches, b) linear extension versus multi-step extension, c) blind backtracking versus informed backtracking, and d) reasoing with inconsistent ontologies versus debugging inconsistent ontologies. For the evaluation, we consider the following three benchmarking factors: quality of query answers, performance, and scalability. We finally discuss the results and draw the conclusions about the future of processing inconsistent ontologies.

This document is organized as follows: Chapter 2 overviews various approaches of processing inconsistent ontologies, which include various methods of reasoning with inconsistent ontologies, and various systems of debugging inconsistent ontologies. Chapter 3 presents a methodology study of benchmarking of processing inconsistent ontologies and introduce the benchmarking suite, which are developed for the task of benchmarking of processing inconsistent ontologies in the KnowledgeWeb project. Chapter 4 provides the detailed report of the benchmarking experiments and make the analysis of the evaluation of various approaches of processing inconsistent ontologies. Chapter 5 discusses further work and concludes the document.

Chapter 2

Processing of Inconsistent Ontologies

In this chapter, we will provide an overview of the framework of reasoning with inconsistent ontologies and various approaches for processing inconsistent ontologies. Moroever, in this chapter we will propose a semantic approach for reasoning with inconsistent ontologies.

2.1 General Framework of Reasoning with Inconsistent Ontologies

A general framework of reasoning with inconsistent ontologies is developed in [Huang *et al.*, 2005][1] , in which relevance based selection functions are used to obtain meaningful answers by using a non-standard reasoning, where the meaningfulness is interpreted as the answer is supported by a selected consistent sub-ontology of the inconsistent ontology, and its negative answer is not supported. The main idea of the framework is: given a selection function, which can be defined on the syntactic or semantic relevance, we select some consistent sub-theory from an inconsistent ontology. Then we apply standard reasoning on the selected sub-theory to find meaningful answers. If a satisfying answer cannot be found, the relevance degree of the selection function is made less restrictive thereby extending the consistent sub-theory for further reasoning.

In the following, we use $\Sigma \models \phi$ to denote that ϕ is a consequence of Σ in the standard reasoning[2], and we will use $\Sigma \mid\!\approx \phi$ to denote that ϕ is a consequence of Σ in the non-standard reasoning. The values of non-standard inference are defined as follows,

[1] More details of the framework can be found in [Huang *et al.*, 2006]

[2] Namely, for any model M of Σ, $M \models \phi$.

following the 4-valued schema by [Belnap, 1977]:

Accepted:	$\Sigma \mid\!\approx \phi$ and $\Sigma \not\mid\!\approx \neg\phi$
Rejected:	$\Sigma \not\mid\!\approx \phi$ and $\Sigma \mid\!\approx \neg\phi$
Overdetermined:	$\Sigma \mid\!\approx \phi$ and $\Sigma \mid\!\approx \neg\phi$
Undetermined:	$\Sigma \not\mid\!\approx \phi$ and $\Sigma \not\mid\!\approx \neg\phi$

Selection functions play the main role in the framework of reasoning with inconsistent ontologies. A system of reasoning with inconsistent ontologies uses a selection function to determine which consistent subsets of an inconsistent ontology should be considered in its reasoning process. The general framework is independent of the particular choice of selection function. The selection function can either be based on a syntactic approach, like Chopra, Parikh, and Wassermann's syntactic relevance [Chopra *et al.*, 2000], or based on semantic relevance like for example in computational linguistics as in Wordnet [Budanitsky and Hirst, 2001] or based on semantic relevance which is measure by the co-occurrence of concepts in search engines like Google.

In the framework of reasoning with inconsistent ontologies, selection functions are designed to query-specific, which is different from the traditional approach in belief revision and nonmonotoic reasoning, which assumes that there exists a general preference ordering on formulas for selection. 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. A selection function s is a mapping $s : \mathcal{P}(\mathbf{L}) \times \mathbf{L} \times N \rightarrow \mathcal{P}(\mathbf{L})$ such that $s(\Sigma, \phi, k) \subseteq \Sigma$. A selection function s is called *monotonic* if the subsets it selects monotonically increase or decrease, i.e., $s(\Sigma, \phi, k) \subseteq s(\Sigma, \phi, k+1)$, or vice verse. For monotonically increasing selection functions, the initial set is either an emptyset, i.e., $s(\Sigma, \phi, 0) = \emptyset$, or a fixed set Σ_0. For monotonically decreasing selection functions, usually the initial set $s(\Sigma, \phi, 0) = \Sigma$. The decreasing selection functions will reduce some formulas from the inconsistent set step by step until they find a maximally consistent set.

Monotonically increasing selection functions have the advantage that they do not have to return *all* subsets for consideration at the same time. If a query $\Sigma \mid\!\approx \phi$ can be answered after considering some consistent subset of the ontology Σ for some value of k, then other subsets (for higher values of k) don't have to be considered any more, because they will not change the answer of the reasoner.

2.2 Strategies

A linear extension strategy is carried out as shown in Figure 2.1. Given a query $\Sigma \mid\!\approx \phi$, the initial consistent subset Σ' is set. Then the selection function is called to return a consistent subset Σ'', which extends Σ', i.e., $\Sigma' \subset \Sigma'' \subseteq \Sigma$ for the linear extension strategy. If the selection function cannot find a consistent superset of Σ', the inconsistency

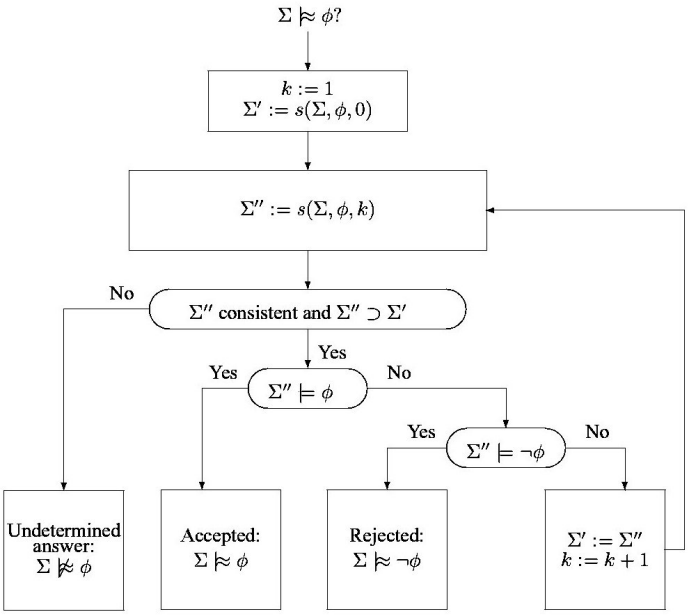

Figure 2.1: Linear Extension Strategy.

reasoner returns the answer 'undetermined' (i.e., unknown) to the query. If the set Σ'' exists, a classical reasoner is used to check if $\Sigma'' \models \phi$ holds. If the answer is 'yes', the reasoner returns the 'accepted' answer $\Sigma \not\approx \phi$. If the answer is 'no', the reasoner further checks the negation of the query $\Sigma'' \models \neg\phi$. If the answer is 'yes', the reasoner returns the 'rejected' answer $\Sigma \not\approx \neg\phi$, otherwise the current result is undetermined, and the whole process is repeated by calling the selection function for the next consistent subset of Σ which extends Σ''.

It is clear that the linear extension strategy may result in too many 'undetermined' answers to queries when the selection function picks the wrong sequence of monotonically increasing subsets. It would therefore be useful to measure the successfulness of (linear) extension strategies. Notice, that this depends on the choice of the monotonic selection function.

In general, one should use an extension strategy that is not over-determined and not undetermined. For the linear extension strategy, we can prove that a reasoner using a linear extension strategy is never over-determined, may be undetermined, always sound, and always meaningful[Huang *et al.*, 2004]. A reasoner using a linear extension strategy is useful to create meaningful and sound answers to queries. It is always locally sound and locally complete with respect to a consistent set Σ_0, if the selection function always starts with the consistent set Σ_0 (i.e., $s(\Sigma, \phi, 0) = \Sigma_0$). Unfortunately it may not be maximal. The advantages of the linear strategy is that the reasoner can always focus on the current working set Σ'[3]. The reasoner doesn't need to keep track of the extension chain. The

[3]Alternatively it is called the selected set.

disadvantage of the linear strategy is that it may lead to an inconsistency reasoner that is undetermined. There exists other strategies which can improve the linear extension approach, for example, by backtracking and heuristics evaluation. We will disuss how it can be achieved in the over-determined processing in Section 2.7.

2.3 Relevance-based Selection Functions

In [Chopra *et al.*, 2000], Chopra, Parikh, and Wassermann propose the syntactic relevance to measure the relationship between two formulas in belief sets, so that the relevance can be used to guide the belief revision based on Schaerf and Cadoli's method of approximate reasoning[Schaerf and Cadoli, 1995]. Given a formula set Σ, two atoms p, q are directly relevant, denoted by $R(p, q, \Sigma)$ iff there is a formula $\alpha \in \Sigma$ such that p, q appear in α. A pair of atoms p and q are k-relevant with respect to Σ iff there exist $p_1, p_2, \ldots, p_k \in \mathcal{L}$ such that: (a) p, p_1 are directly relevant; (b) p_i, p_{i+1} are directly relevant, $i = 1, \ldots, k-1$; and (c) p_k, q are directly relevant (i.e., directly relevant is k-relevant for $k = 0$).

The notions of relevance above are based on propositional logics. However, ontology languages are usually written in some fragment of first order logic. We extend the ideas of relevance to description logic-based ontology language. The Direct relevance between two formulas are defined as a binary relation on formulas, namely $\mathcal{R} \subseteq L \times L$. Given a direct 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:

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

Namely, a formula ϕ is relevant to a formula set Σ iff there exists a formula $\psi \in \Sigma$ such that ϕ and ψ are directly relevant. We can similarly specialize the notion of k-relevance. Two formulas ϕ, ϕ' are k-relevant with respect to a formula set Σ iff there exist formulas $\psi_0, \ldots \psi_k \in \Sigma$ such that ϕ and ψ_0, ψ_0 and ψ_1, ..., and ψ_k and ϕ' are directly relevant. A formula ϕ is k-relevant to a formula set Σ iff there exists a formula $\psi \in \Sigma$ such that ϕ and ψ are k-relevant with respect to Σ.

We can use a relevance relation to define a selection function s to extend the query '$\Sigma \mid\!\approx \phi$?' as follows: We start with the query formula ϕ as a starting point for the selection based on syntactic relevance. Namely, we define:

$$s(\Sigma, \phi, 0) = \emptyset.$$

Then the selection function selects the formulas $\psi \in \Sigma$ which are directly relevant to ϕ as a working set (i.e. $k = 1$) to see whether or not they are sufficient to give an answer to the query. Namely, we define:

$$s(\Sigma, \phi, 1) = \{\psi \in \Sigma \mid \phi \text{ and } \psi \text{ are directly relevant}\}.$$

If the reasoning process can obtain an answer to the query, it stops. Otherwise the selection function increases the relevance degree by 1, thereby adding more formulas that are relevant to the current working set. Namely, we have:

$$s(\Sigma, \phi, k) = \{\psi \in \Sigma \mid \psi \text{ is directly relevant to } s(\Sigma, \phi, k-1)\},$$

for $k > 1$. This leads to a "fan out" behavior of the selection function: the first selection is the set of all formulae that are directly relevant to the query; then all formulae are selected that are directly relevant to that set, etc. This intuition is formalized in this: The syntactic relevance-based selection function s is monotonically increasing. We observe that If $k \geq 1$, then

$$s(\Sigma, \phi, k) = \{\phi | \phi \text{ is (k-1)-relevant to } \Sigma\}$$

The syntactic relevance-based selection functions defined above usually grows up to an inconsistent set rapidly. That may lead to too many undetermined answers. In order to improve it, we can require that the selection function returns a consistent subset Σ'' at the step k when $s(\Sigma, \phi, k)$ is inconsistent such that $s(\Sigma, \phi, k-1) \subset \Sigma'' \subset s(\Sigma, \phi, k)$. It is actually a kind of backtracking strategies which are used to reduce the number of undetermined answers to improve the linear extension strategy. We call the procedure an *over-determined processing*(ODP) of the selection function. Note that the over-determined processing does not need to exhaust the powerset of the set $s(\Sigma, \phi, k) - s(\Sigma, \phi, k-1)$, because of the fact that if a consistent set S cannot prove or disprove a query, then nor can any subset of S. Therefore, one approach of ODP is to return just a maximally consistent subset. Let n be $|\Sigma|$ and k be $n - |S|$, i.e., the cardinality difference between the ontology Σ and its maximal consistent subset S (note that k is usually very small), and let C be the complexity of the consistency checking. The complexity of the over-determined processing is polynomial to the complexity of the consistency checking. Note that ODP introduces a degree of non-determinism: selecting different maximal consistent subsets of $s(\Sigma, \phi, k)$ may yield different answers to the query $\Sigma \mid\!\approx \phi$. The simplest example of this is $\Sigma = \{\phi, \neg\phi\}$.

2.4 Syntactic Relevance-based Selection Functions

There are various ways to define syntactic relevance between two formulas. Given a formula ϕ, we use $I(\phi), C(\phi), R(\phi)$ to denote the sets of individual names, concept names, and relation names that appear in the formula ϕ respectively. In [Huang *et al.*, 2005], we propose a direct relevance which considers the syntactic existence of a common concept/role/individual name in two formulas.

Two formula ϕ, ψ are directly syntactic relevant, written $\mathcal{R}_{SynRel}(\phi, \psi)$, iff there is a common name which appears both in formula ϕ and formula ψ, i.e.,

$$\langle \phi, \psi \rangle \in \mathcal{R}_{SynRel} \text{ iff } I(\phi) \cap I(\psi) \neq \emptyset \vee C(\phi) \cap C(\psi) \neq \emptyset \vee R(\phi) \cap R(\psi) \neq \emptyset.$$

In [Huang and van Harmelen, 2006] we propose directly concept relevance, as a variant of directly syntactic relevance, which considers only one side of concept names in subsumption axioms to track along the concept hierarchy in an ontology more efficiently. For a query $C1 \sqsubseteq C2$, we start from C_1, i.e., the left hand side of the query to select a set of relevant axioms to see whether or not the selected set entails the query[4]. If 'not', the selection function will extend the selected set by adding more formulas in the ontology which are relevant to the current selected set until the query is proved or disproved, or 'undetermined' when there is no more relevant formulas, as it is described in the extension strategies.

Thus, a formula ϕ is directly concept-relevant to a formula ψ, written $R_{SynConRel}(\phi, \psi)$, iff
(i) $C(C_1) \cap C(\psi)) \neq \emptyset$ if the formula ϕ has the form $C_1 \sqsubseteq C_2$,
(ii) $C(C_1) \cap C(\psi) \neq \emptyset$ or $C(C_2) \cap C(\psi) \neq \emptyset$ if the formula ϕ has the form $C_1 = C_2$.
(iii) $C(C_1) \cap C(\psi) \neq \emptyset$ or $\ldots$ or $C(C_n) \cap C(\psi) \neq \emptyset$ if the formula ϕ has the form $disjoint(C_1, \ldots, C_n)$.

In [Huang and van Harmelen, 2006, Huang *et al.*, 2005], a preliminary evaluation of the prototype by applying it to several inconsistent ontologies is reported. The syntactic relevance approach works with real-world inconsistent ontologies because it mimics our intuition that real- world truth is (generally) preserved best by the argument with the shortest number of steps; and whatever process our intuitive reasoning uses, it is very likely that it would somehow privilege just these shortest path arguments. However, the problem is that the syntactic relevance approach requires that knowledge engineers have to carefully specify their ontologies to represent their intuitive understandings on the knowledge. The syntactic relevance approach require that knowledge engineers carefully specify ontologies based on the intuition to balance reasoning paths. The semantic approach is developed to be expected to be a complement of syntactic approaches. We will propose a semantic approach for reasoning with inconsistent ontologies in Section 2.6.

2.5 K-extension

Compared with the linear extension strategy, the k-extension can be considered as a multi-step extension strategy. The k-extension strategy allows for k-step difference on the paths. Namely, if the step difference between two paths is less than k, they are considered equally. We call the step difference number k *the k-value* of the algorithm. Thus, the linear extension is actually 1-extension, namely its k-value is 1. The general idea of k-extension is that the algorithm considers k steps in the linear extension as a single step in the k-extension, by which the algorithm can tolerate k-step unbalanced reasoning paths.

The algorithm of k-extension is shown in Figure 2.2. In the algorithm, we use Ks to

[4]PION2 supports only subsumption queries and instance queries. Instance queries are handled by their corresponding subsumption queries internally.

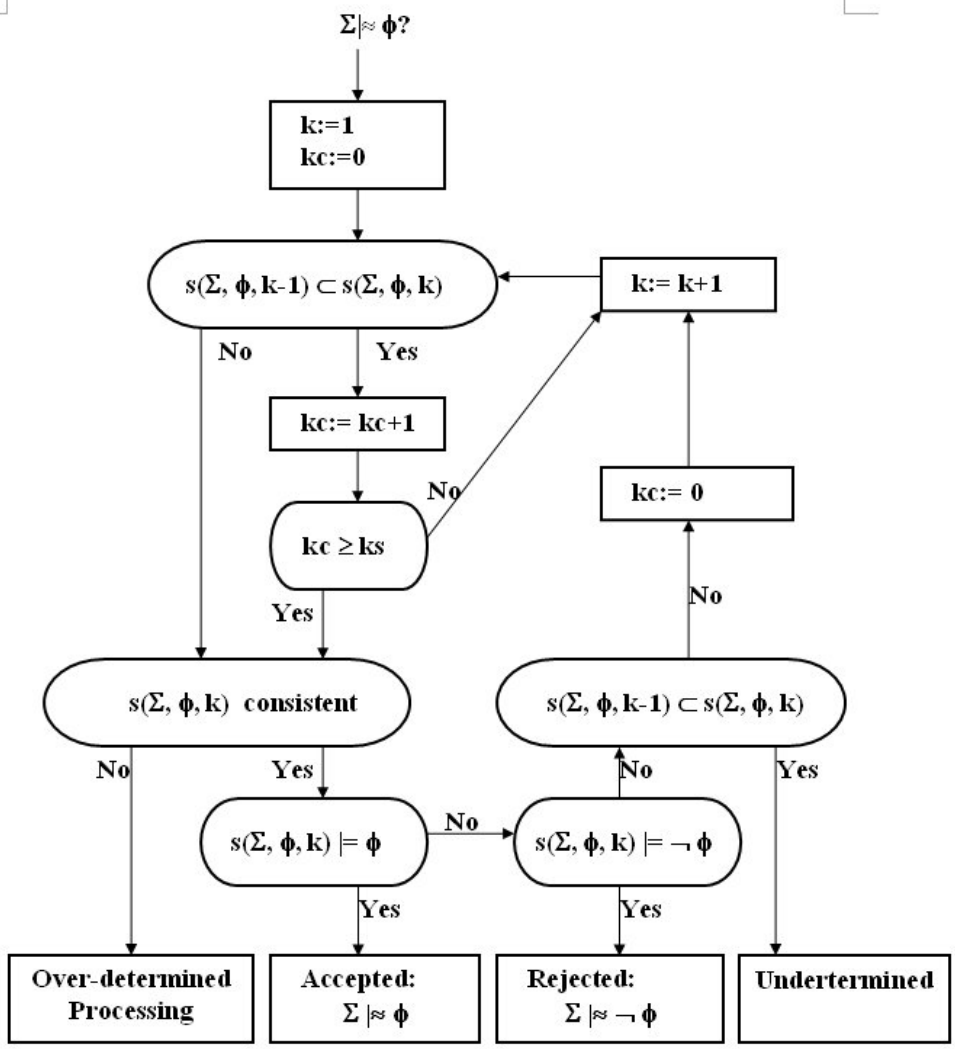

Figure 2.2: k Extension Strategy.

denote the k-value of the algorithm, Σ to denote the ontology, and ϕ to denote the query respectively. We use two additional counters K and Kc in the extension processing. The former is a global counter which counts how many steps have been done in the extension, which plays the same role like the counter k in the algorithm of the linear extension. The latter is a local counter that counts how many steps have been done since the last k-extension is done. The algorithm starts its selection function $s(\Sigma, \phi, 0)$ with the global counter K with its initial value $K = 1$ and the local counter with its initial value $Kc = 0$. If there are no new selected formulas can be found, i.e., it is not the case that $s(\Sigma, \phi, K - 1) \subset s(\Sigma, \phi, K)$, then the algorithm takes the current extension set as the selected set. When the local counter Kc is bigger than the k-value Ks, then the algorithm takes the current extension set as the selected set. Namely, the algorithm stops the k-extension. It checks whether or not the selected set $s(\Sigma, \phi, K)$ is consistent. If the selected set is inconsistent, the algorithm calls the over-determined processing. If the selected set is consistent, then the algorithm checks whether or not the query or its negation is entailed by the selected set. If none of the query and its negation can be proved by the selected set, then the algorithm will start a new Ks step extension, by setting the local counter Kc with its initial value $Kc = 0$. If none of the query and its negation can be proved by the selected set and there are no new selected formulas in the k-extension, the algorithm will stops with the answer 'undetermined'.

For the k-extension algorithm, it is easy to see that when $Ks > 1$, i.e., non linear extension, k-extension would more easily get to the over-determined processing, because it considers multiple extension steps as a single extension step like that in the linear extension algorithm. That required us to consider different strategies of over-determined processing. We will discuss variants of over-determined processing strategies in Section 2.7.

2.6 Semantic Relevance Based Selection Functions

The syntactic relevance-based selection functions prefer shorter paths to longer paths in the reasoning. It requires knowledge engineers should carefully design ontologies to avoid unbalanced reasoning path. The k-extension can tolerate the k-step difference of reasoning paths. Setting the k-value as a very big number can make the reasoning processing tolerates big step difference in reasoning paths. However, it would lead to another big problem that it encounters the over-determined processing problem much more easily.

Naturally we will consider semantic relevance based selection functions as alternatives of syntactic relevance based selection functions. In the following, we will propose a semantic relevance based section function that is developed based on Google distance. Namely, we want to take advantage of the vast knowledge on the web by using Google based relevance measure, by which we can obtain light-weight semantics for selection functions. The basic assumption here is that: more frequently two concepts appear in the same web page, more semantically relevant they are, because most

of web pages are meaningful texts. Therefore, information provided by a search engine can be used for the measurement of semantic relevance among concepts. We select Google as the targeted search engine, because it is the most popular search engine nowaday. The second reason why we select Google is that Google distances are well studied in [Cilibrasi and Vitanyi, 2004, Cilibrasi and Vitany, 2007].

In [Cilibrasi and Vitanyi, 2004, Cilibrasi and Vitany, 2007], Google Distances are used to measure the co-occurrence of two keywords over the Web. Normalized Google Distance (NGD) is introduced to measure semantic distance between two concepts by the following definition:

$$NGD(x,y) = \frac{max\{logf(x), logf(y)\} - logf(x,y)}{logM - min\{logf(x), logf(y)\}}$$

where

$f(x)$ is the number of Google hits for the search term x,

$f(y)$ is the number of Google hits for the search term y,

$f(x,y)$ is the number of Google hits for the tuple of search terms x and y, and,

M is the number of web pages indexed by Google[5].

$NGD(x,y)$ can be understood intuitively as a measure for the symmetric conditional probability of co-occurrence of the search terms x and y.

$NGD(x,y)$ usally takes a real number between 0 and 1[6]. $NGD(x,x) = 0$ means that any search item is always the closest to itself. $NGD(x,y)$ is defined for two search items x and y, which measures the semantic dissimilarity, alternatively called *semantic distance*, between them.

Semantic relevance is considered as the reverse relation of semantic dissimilarity: the more semantically relevant two concepts are, the smaller the distance between them. Assuming that both relevance and distance are taken from the [0,1] interval, this boils down to

$$Similarity(x,y) = 1 - Distance(x,y).$$

In the following we use the terminologies *semantic dissimilarity* and *semantic distance* interchangeably.

To use semantic dissimilarity for reasoning with inconsistent ontologies, we define the dissimilarity measure between two formulas in terms of the dissimilarity measure between two concepts/roles/individuals from the two formulas. Moreover, in the following we consider only concept names $C(\phi)$ as the symbol set of a formula ϕ to simplify the formal definitions. However, note that the definitions can be easily generalised into ones

[5]Currently, the Google search engine indexs approximately tenbillion pages.

[6]$NGD(x,y)$ may exceed 1 for some natural terms. NGD values greater than 1 may be thougt to correspond to the idea of negative correlation in probability theory. See [Cilibrasi and Vitanyi, 2004] for details.

in which the symbol sets contain also roles and individuals. We use $SD(\phi, \psi)$ to denote the semantic distance between two formulas. We expect the semantic distance between two formulas $SD(\phi, \psi)$ to satisfy the following intuitive properties:

- **Range** The semantic distance is a real number between 0 and 1: $0 \leq SD(\phi, \psi) \leq 1$ for any ϕ and ψ.
- **Reflexivity** Any formula is always semantically closest to itself: $SD(\phi, \phi) = 0$ for any ϕ.
- **Symmetry** The semantic distance between two formulas is symmetric: $SD(\phi, \psi) = SD(\psi, \phi)$ for any ϕ and ψ.
- **Maximum distance** If all symbols in a formula are semantically most-dissimilar from any symbol of another formula, then the two formulas are totally dissimilar: if $SD(C_i, C_j) = 1$ for all $C_i \in C(\phi)$ and $C_j \in C(\psi)$, then $SD(\phi, \psi) = 1$.
- **Intermediate values** If some symbols are shared between two formula, and some symbols are semantically dissimilar, the semantic distance between the two formulas is neither minimal nor maximal:
  If $C(\phi) \cap C(\psi) \neq \emptyset$ and $C(\phi) \not\subseteq C(\psi)$ and $C(\psi) \not\subseteq C(\phi)$ then $0 < SD(\phi, \psi) < 1$.

However, note that semantic distances do not always satisfy the Triangle Inequality

$$SD(\phi, \psi) + SD(\psi, \rho) \geq SD(\phi, \rho),$$

a basic property of distances in a metric topology. [Lin, 1998] provides a counter-example of the Triangle Inequality for a semantic similarity measure.

$NGD(x, y)$ is defined between two search items x and y. Simple ways to extend this to measure the semantic distance between two formulas are to take either the minimal, the maximal or the average NGD values between two concepts (or roles, or individuals) which appear in two formulas as follows:

$$\begin{aligned}
&SD_{min}(\phi, \psi) = \\
&\quad min\{NGD(C_i, C_j) | C_i \in C(\phi) \; and \; C_j \in C(\psi)\} \\
&SD_{max}(\phi, \psi) = \\
&\quad max\{NGD(C_i, C_j) | C_i \in C(\phi) \; and \; C_j \in C(\psi)\} \\
&SD_{ave}(\phi, \psi) = \\
&\quad \frac{sum\{NGD(C_i, C_j) | C_i \in C(\phi) \; and \; C_j \in C(\psi)\}}{(|C(\phi)| * |C(\psi)|)}
\end{aligned}$$

where $|C(\phi)|$ means the cardinality of $C(\phi)$. However, it is easy to see that SD_{min} and SD_{max} do not satisfy the Intermediate Values property, and SD_{ave} does not satisfy Reflexivity and the Maximum Distance property.

We therefore propose a semantic distance which is measured by the ratio of the summed distance of the difference between two formulas to the maximal distance between two formulas:

Definition 1 (Semantic Distance)

$$SD(\phi,\psi) = \frac{sum\{NGD(C_i,C_j)|C_i \in C(\phi)\backslash C(\psi), C_j \in C(\psi)\backslash C(\phi)\}}{(|C(\phi)| * |C(\psi)|)}$$

The intuition behind this definition is to sum the semantic distances between all terms that are not shared between the two formulae, but these must be normalised (dividing by the maximum distance possible) to bring the value back to the [0,1] interval.

It is easy to prove the following:

Proposition 2 The semantic distance $SD(\phi,\psi)$ satisfies the properties Range, Reflexivity, Symmetry, Maximum Distance, and Intermediate Values.

Using the semantic distance defined above, the obvious way to define a relevance relation for selection functions in reasoning with inconsistent ontologies is to take the semantically closest formulas as directly relevant:

$$\langle \phi,\psi\rangle \in \mathcal{R}_{sd} \text{ iff} \neg\exists\psi' \in \Sigma : SD(\phi,\psi') < SD(\phi,\psi).$$

(i.e. there exist no other formulas in the ontology that is semantically closer)

Given this semantic relevance relation, we now need to define a selection function. In the syntactic approach, we used the query formula as the starting point for the selection function. Since we intend to use the selection function for reasoning over ontologies, we propose a specific approach to deal with subsumption queries, which are of the form $C_1 \sqsubseteq D$ where C_1 is a concept. In this new approach, C_1 is considered as the central concept of the query, and the newly defined selection function will track along the concept hierarchy in an ontology and always add to the selected set the closest formulas to C_1 which have not yet been selected[7]:

Definition 3 (Semantic Selection Function)

$$\begin{aligned} &s(\Sigma, C_1 \sqsubseteq D, 0) = \emptyset \\ &s(\Sigma, C_1 \sqsubseteq D, k) = \\ &\quad s(\Sigma, C_1 \sqsubseteq D, k-1) \cup \\ &\quad \{\psi \in \Sigma \mid \neg\exists\psi' \in \Sigma : \ \psi' \notin s(\Sigma, C_1 \sqsubseteq D, k-1) \wedge \\ &\qquad\qquad\qquad\qquad SD(\psi', C_1) < SD(\psi, C_1)\} \end{aligned}$$

In words: the k-th selected set equals the previously selected set (at $k-1$), augmented with the semantically closest formula that wasn't selected yet in the previous round.

[7]It is easy to see the definition about $SD(\phi,\psi)$ is easily extended into a definition about $SD(\phi,C)$, where ϕ,ψ are formulas, and C is a concept. Moreover, it is easy to see that $SD(\phi_1,C) < SD(\phi_2,C)$ iff $NGD(D_1,C) < NGD(D_2,C)$ for any ϕ_1 is of the form $C_1 \sqsubseteq D_1$ and any ϕ_2 is of the form $C_1 \sqsubseteq D_2$ where C, C_1, D_1 and D_2 are different concepts.

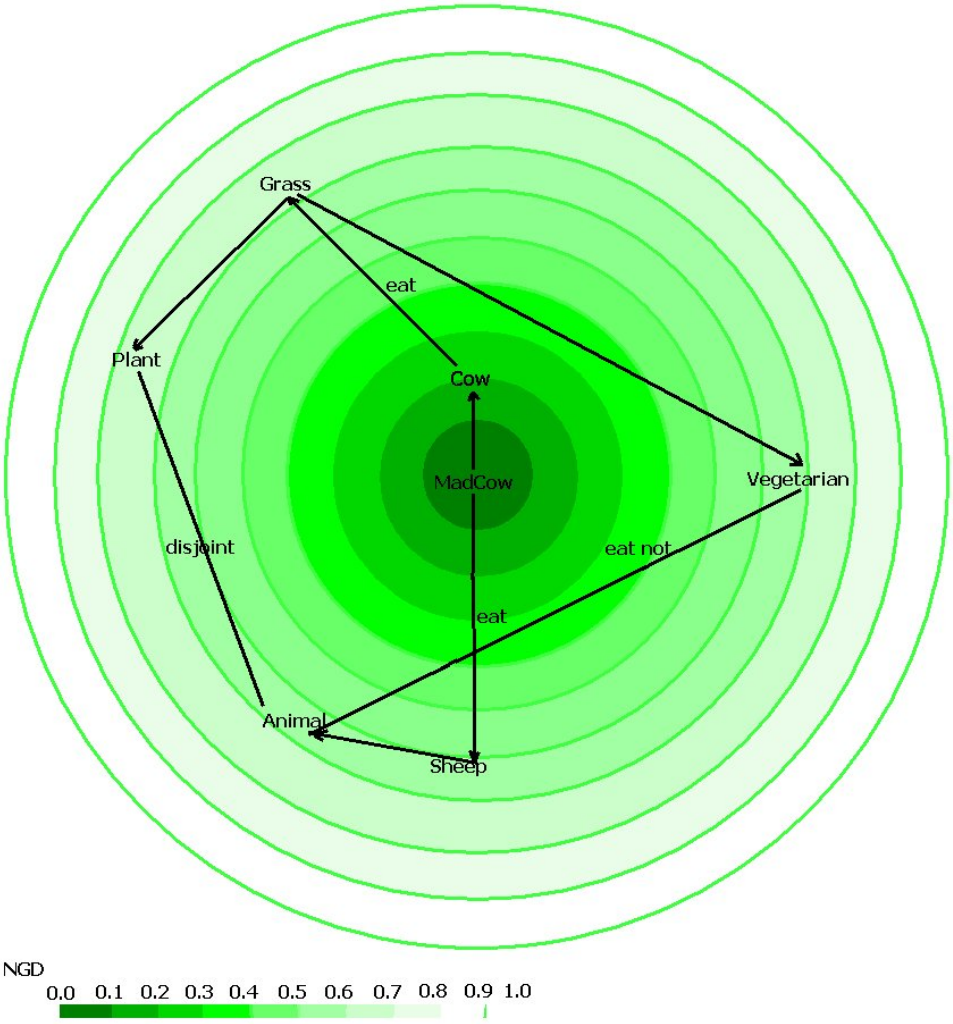

Figure 2.3: NGD and MadCow Queries

Example: Consider the MadCow ontology in which Cows are speficied as Vegetarians and MadCows are specified as Cows which eat brains of sheep. In this ontology, the concept 'MadCow'is unsatisfiable, because it belongs to a parent concept 'Vegetarian' and its negative concept 'non-Vegetarian'. An improved specification of the MadCow ontology is to introduce balanced paths in the specification as follows:

$MadCow - Cow - eat.Grass - Grass\ is\ not\ animal - Vegetarian.$

$MadCow - eat.Sheep - Sheep\ are\ animals - eat.Animals - not\ Vegetarian.$

However, it still fails in the syntactic distance. Figure 2.3 shows how the semantic distance is used to obtain intuitive answers on the MadCow ontology. By calculation of the Normalised Google Distance, we know that

$$NGD(MadCow, Grass) = 0.722911$$
$$NGD(MadCow, Sheep) = 0.612001.$$

Hence, the semantic distance between MadCow and Sheep is shorter than the semantic distance between MadCow and Grass (even though their syntactic distance is larger). Because of this, the reasoning path between *MadCow* and *Sheep* is preferred to the reasoning path between *MadCow* and *Grass*. Thus, we obtain the intuitive answer that *MadCow are not Vegetarians* instead of the previously obtained counter-intuitive answer that *MadCow are Vegetarians*. The intuition here is that although syntactically, the *MadCow - Sheep* path is the longer of the two, the accumulated semantic distance on this syntactically longer path is still shorter than the semantic distance on the syntactically short *MadCow - Grass* path.

2.7 Variants of Over-determined Processing

The reasoning extension procedure usually grows up to an inconsistent set rapidly. That may lead to too many undetermined answers. In order to improve it, over-determined processing is introduced, by which we require that the selection function returns a consistent subset Σ'' at the step k when $s(\Sigma, \phi, k)$ is inconsistent such that $s(\Sigma, \phi, k-1) \subset \Sigma'' \subset s(\Sigma, \phi, k)$. It is actually a kind of backtracking strategies used to reduce the number of undetermined answers to improve the extension strategy. An easy solution to the over-determined processing is to return the first maximal consistent subset (FMC) of $s(\Sigma, \phi, k)$, based on certain search procedure. Query answers which are obtained by this procedure are still sound, because they are supported by a consistent subset of the ontology. However, it does not always provide intuitive answers because it depends on the search procedure of maximal consistent subset in over-determined processing. In the following we will propose a semantic relevance based approach for over-determined processing. Before doing that, we would like to discuss the pros and cons of semantic relevance first.

2.7.1 Pros and Cons of Semantic Relevance

Although empirical findings will only be discussed in Chapter 4, we can already establish some of the advantages and disadvantages of the semantic approach to relevance.

Slower fan out behavior: As is clear from the definition about the semantic selection function, the growth of a relevance based on semantic distance is much slower than one based on syntactic relevance. In fact, at each step the semantic relevance set grows by a single formula (barring the exceptional case when some formulas share the same distance to the query).

Almost never needs a backtracking step: This slower growth of semantic relevance means that it will also hardly ever need a backtracking step, since the relevance set is unlikely to become "too large" and inconsistent.

Expensive to compute: Again by inspecting the definition about the semantic selection function, it is clear that computing the semantic relevance is expensive: it requires to know the semantic distance between the query and *every* formula ψ in the theory Σ. Furthermore, this must be done again for every new query concept C_1. With realistic modern ontologies often at a size of $O(10^5)$ concepts, and a computation time in the order of 0.2 secs for a single NGD-value, this would add a prohibitive cost to each query[8].

The picture that emerges from the pro's and cons of the semantic approach is that syntactic relevance is cheap to compute, but grows too quickly and then has to rely on a blind backtracking step, while semantic relevance has controlled growth, with no need for backtracking, but is expensive to compute.

2.7.2 Mixed Approach for Over-determined Processing

In this section, we will propose a *mixed* approach which combine the advantages of both: we will use a syntactic-relevance selection function to grow the selection set cheaply, but we will use semantic relevance to improve the backtracking step.

Instead of picking the first maximal consistent subset through a blind breadth-first descent, we can prune semantically less relevant paths to obtain a consistent set. This is done by removing the most dissimilar formulas from the set $s(\Sigma, \phi, k) - s(\Sigma, \phi, k-1)$ first, until we find a consistent set such that the query ϕ can be proved or disproved.

[8] although some of this can be amortised over multiple queries by caching parts of the values that make up the NGD.

Example: Taking the same example of the MadCow ontology above, we can see from Figure 2.3 that the path between *MadCow* and *Grass* can be pruned first, rather than pruning the path between *MadCow* and *Sheep*, because of the NGD information:

$$NGD(MadCow, Sheep) = 0.612001$$
$$NGD(MadCow, Grass) = 0.722911$$

Thus, the path *MadCow - Grass* (which lead to the counter-intuitive conclusion that *MadCow are vegetarians*) is pruned first.

We call this *over-determined processing (ODP) using path pruning with Google distance*. While syntactic over-determined processing can be seen as a blind breadth-first search, semantic-relevance ODP can be seen as a hill-climbing procedure, with the semantic distance as the heuristic.

Notice that semantic backtracking is not guaranteed to yield a *maximal* consistent subset. Consequently, the completeness of the algorithm may be affected, since we might have removed too many formulas from the relevance set in our attempt to restore consistency, thereby loosing the required implication to obtain the intuitive answer. Furthermore, it is possible that the semantic backtracking might lead to the *wrong* consistent subset, one supporting ϕ where $\neg\phi$ would have been the intuitive answer, or vice versa. In our experiment in Chapter 4 we will find that indeed the completeness drops (as expected), but not by very much, while the unsoundness does not increase at all (making us belief that SD is a good heuristic for the hill-climbing search towards a consistent subset).

Finally, the semantic distance provides the possibility for adjustable behaviour of the backtracking increments that are taken in the over-determined processing phase. We introduce a cutting level α ($0 \leq \alpha \leq 1$), and instead of only pruning the semantically least relevant paths one by one until we obtain a consistent subset, we now prune in one step all formulas whose distance to the query is higher than α. In this way, α plays the role of a threshold, so that the processing can be sped up by pruning in a single step all those formulas which do not meet the relevance threshold. This might of course increase the amount of undetermined answers (since we may have over-pruned), but it allows us to make a tradeoff between the amount of undetermined answers and the time performance. In Chapter 4 we will report an experiment in which this tradeoff obtains a 500% efficiency gain in exchange for only a 15.7% increase in undetermined answers.

2.8 Debugging Incoherent Terminologies

Description Logics are a family of well-studied set-description languages which have been in use for over two decades to formalize knowledge. They have a well-defined model theoretic semantics, which allows for the automation of a number of reasoning services. In this section we provide an overview on various methods for debugging inconsistent ontologies.

2.8.1 Logical errors in Description Logic terminologies

For a detailed introduction to Description Logics we point to the second chapter of the DL handbook [Baader *et al.*, 2003]. Briefly, in DL concepts will be interpreted as subsets of a domain, and roles as binary relations. Let, throughout the paper, $\mathcal{T} = \{ax_1, \ldots, ax_n\}$ be a set of (terminological) axioms, where ax_i is of the form $C_i \sqsubseteq D_i$ for each $1 \leq i \leq n$ and arbitrary concepts C_i and D_i. We will also use terminological axioms of the form $C = D$ and disjointness statements *disjoint(C,D)* between two concepts C and D, which are simple abbreviations of $C \sqsubseteq D \& D \sqsubseteq C$, and $C \sqsubseteq \neg D$ respectively. Most DL systems also allow for assertional axioms in a so-called ABox. In this paper, ABoxes will not be considered. Throughout the paper the term *ontologies* will refer to general knowledge bases which possibly include both terminological and assertional knowledge. The term *terminology* is solely used in the technical sense of a DL TBox.

2.8.1.1 Unsatisfiability and Incoherence

Let $\mathcal{U}$ be a finite set of objects, called the universe. A mapping $\mathcal{I}$, which interprets DL concepts as subsets of $\mathcal{U}$ is a *model* of a terminological axiom $C \sqsubseteq D$, 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 these 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 C is *unsatisfiable* w.r.t. a terminology $\mathcal{T}$ if, and only if, $C^{\mathcal{I}} = \varnothing$ for all models $\mathcal{I}$ of $\mathcal{T}$.

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

Conceptually, these cases often point to modeling errors because we assume that a knowledge modeler would not specify something like an impossible concept in a complex way.

Table 2.1 demonstrates this principle. Consider the (incoherent) TBox $\mathcal{T}_1$, where A, B and C, as well as $A_1, \ldots, A_7$ are concept names, and r and s roles. Satisfiability testing returns a set of unsatisfiable concept names $\{A_1, A_3, A_6, A_7\}$. Although this is still of manageable size, it hides crucial information, e.g., that unsatisfiability of A_1 depends, among others, on unsatisfiability of A_3, which is in turn unsatisfiable because of the contradictions between A_4 and A_5. We will use this example later in this paper to explain our debugging methods.

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

Table 2.1: A small (incoherent) TBox $\mathcal{T}_1$

2.8.1.2 Unfoldable $\mathcal{ALC}$ TBoxes

In this section we study ways of *debugging and diagnosing* of incoherence and unsatisfiability in DL terminologies. The general ideas can easily be extended to inconsistency of ontologies with assertions as suggested in [Schlobach *et al.*, 2006]. As the evaluation in this document will be about terminological debugging only, we will restrict the technical definitions to the necessary notions.

Whereas the definitions of debugging were independent of the choice of a particular Description Logic, we will later present algorithms for the Description Logic $\mathcal{ALC}$, and unfoldable TBoxes, in particular.

$\mathcal{ALC}$ is a simple yet relatively expressive DL with conjunction $(C \sqcap D)$, disjunction $(C \sqcup D)$, negation $(\neg C)$ and universal $(\forall r.C)$ and existential quantification $(\exists r.C)$, where the interpretation function is extended to the different language constructs as follows:

$$
\begin{aligned}
&(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
&(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
&(\neg C)^{\mathcal{I}} = \mathcal{U} \setminus C^{\mathcal{I}} \\
&(\exists R.C)^{\mathcal{I}} = \{d \in \mathcal{U} \mid \exists e \in \mathcal{U} : (d,e) \in R^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\} \\
&(\forall R.C)^{\mathcal{I}} = \{d \in \mathcal{U} \mid \forall e \in \mathcal{U} : (d,e) \in R^{\mathcal{I}} \text{ implies } e \in C^{\mathcal{I}}\}
\end{aligned}
$$

A TBox is called *unfoldable* if the left-hand sides of the axioms (the defined concepts) are atomic and unique, and if the right-hand sides (the definitions) contain no direct or indirect reference to the defined concept [Nebel, 1990]. In T_1, our example TBox, $A_1, \ldots, A_7$ are defined concepts.

2.8.2 Framework for debugging and diagnosis

A theory of debugging and diagnosis and link it to description logic-based systems in [Schlobach and Huang, 2007]. In this case a diagnosis is a smallest set of axioms that needs to be removed or corrected to render a specific concept or all concepts satisfiable.

In some situations, terminologies can contain a large number of unsatisfiable concepts. This can occur for example when terminologies are the result of a merging process of separately developed terminologies, or when closure axioms (i.e. disjointness statements and universal restrictions) are added to terminologies. Unsatisfiability propagates, i.e. one

unsatisfiable concept may cause many other concepts to become unsatisfiable as well. As it is often not clear to a modeler what concepts are the root cause of unsatisfiability, we also describe a number of heuristics that help to indicate reasonable starting points for debugging an terminology.

2.8.2.1 Model-based Diagnosis

The literature on model-based diagnosis is manifold, but we focus on the seminal work of Reiter [Reiter, 1987], and [Greiner *et al.*, 1989], which corrects a small bug in Reiter's original algorithm. We refer the interested reader to a good overview in [Console and Dressler, 1999].

Reiter introduces a diagnosis of a system as the smallest set of components from that system with the following property: the assumption that each of these components is faulty (together with the assumption that all other components are behaving correctly) is consistent with the system description and the observed behavior. In other words: assuming correctness of any one of the components in a diagnosis would cause inconsistency between the system description and the observed behavior. For example, a simple electrical circuit can be defined, consisting of a number of adders. Based on the description of the system and some input values, one can calculate the output of the system. If the observed output is different from the expected output, at least one of the components must be faulty, and diagnoses determine which components could have caused the error.

To apply this definition to a description logic terminology, we regard the terminology as the *system* to be diagnosed, and the axioms as the *components* of this system. The concepts and roles in a concept definition are regarded as *input values*, and the defined concepts as *output values*. If we look at the example terminology from Table 2.1, the *system description* states that it is coherent (i.e. all concepts are satisfiable), but the *observation* is that A_1, A_3, A_6, and A_7 are unsatisfiable. In Reiter's terminology, a minimal set of axioms that need to be removed (or better fixed) is called a diagnosis. This adaptation of Reiter's method leads to the following definition of diagnosis.

Definition 4 Let $\mathcal{T}$ be an incoherent terminology. A *diagnosis for the incoherence problem of* $\mathcal{T}$ is a minimal set of axioms $\mathcal{T}' \subseteq \mathcal{T}$ such that $\mathcal{T} \setminus \mathcal{T}'$ is coherent. Similarly, a *diagnosis for unsatisfiability of a single concept* A in $\mathcal{T}$ is a minimal subset $\mathcal{T}' \subseteq \mathcal{T}$, such that A is satisfiable w.r.t. $\mathcal{T} \setminus \mathcal{T}'$.

Reiter provides a generic method to calculate diagnoses on the basis of conflict sets and their minimal hitting sets. A conflict set is a set of components that, when assumed to be fault free, lead to an inconsistency between the system description and observations. A conflict set is minimal if and only if no proper subset of it is a conflict set. The minimal conflict sets (w.r.t. coherence) for the system in Table 2.1 are $\{ax_1, ax_2\}$, $\{ax_3, ax_4, ax_5\}$, and $\{ax_4, ax_7\}$.

A hitting set H for a collection of sets C is a set that contains at least one element of

each of the sets in C. Formally: $H \subseteq \bigcup_{S \in C} S$ such that $H \cap S \neq \emptyset$ for each $S \in C$. A hitting set is minimal if and only if no proper subset of it is a hitting set. Given the conflict sets above, the minimal hitting sets are: $\{ax_1, ax_3, ax_7\}$, $\{ax_1, ax_4\}$, $\{ax_1, ax_5, ax_7\}$, $\{ax_2, ax_3, ax_7\}$, $\{ax_2, ax_4\}$, and $\{ax_2, ax_5, ax_7\}$.

Reiter shows that the set of diagnoses actually corresponds to the collection of minimal hitting sets for the minimal conflict sets. Hence, the minimal hitting sets given above determine the diagnoses for the system w.r.t. coherence. In [de Kleer and Williams, 1987] diagnosis is extended by providing a method for computing the probabilities of failure of various components based on given measurements. Especially in cases where there are many diagnoses, additional observations (measurements) need to be made in order to determine the actually failing components. The method provided can also determine what observation has the highest discriminating power, i.e. needs to be performed to maximally reduce the number of diagnoses.

2.8.2.2 Debugging

As previously mentioned, the theory of diagnosis is built on minimal conflict sets. But in the application of diagnosis of erroneous terminologies, these minimal conflict sets play a role of their own, as they are the prime tools for debugging, i.e. for the identification of potential errors. For different kind of logical contradictions we introduce several different notions based on conflict sets, the MUPS for unsatisfiability of a concept, the MIPS for incoherence of a terminology.

Minimal unsatisfiability-preserving sub-TBoxes (MUPS)
In [Schlobach and Cornet, 2003b] we introduced the notion of Minimal Unsatisfiability Preserving Sub-TBoxes (MUPS) to denote minimal conflict sets. 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 5 A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a *minimal unsatisfiability preserving sub-TBox (MUPS)* for A in $\mathcal{T}$ 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 $mups(\mathcal{T}, \text{A})$. MUPS for our example TBox $\mathcal{T}_1$ and its unsatisfiable concepts are:

$mups(\mathcal{T}_1, A_1)$: $\{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\}$

$mups(\mathcal{T}_1, A_3)$: $\{\{ax_3, ax_4, ax_5\}\}$

$mups(\mathcal{T}_1, A_6)$: $\{\{ax_1, ax_2, ax_4, ax_6\}, \{ax_1, ax_3, ax_4, ax_5, ax_6\}\}$

$mups(\mathcal{T}_1, A_7)$: $\{\{ax_4, ax_7\}\}$

In the terminology of Reiter's diagnosis each $mups(\mathcal{T}, A)$ is a collection of minimal conflict sets w.r.t. satisfiability of concept A in TBox $\mathcal{T}$.

Remember that a diagnosis is a minimal hitting set for a collection of conflict sets. Hence, from the MUPS, we can also calculate the diagnoses for unsatisfiability of concept A in TBox $\mathcal{T}$, which we will denote $\Delta_{\mathcal{T},A}$.

$\Delta_{\mathcal{T}_1,A1} : \{\{ax_1\}, \{ax_2, ax_3\}, \{ax_2, ax_4\}, \{ax_2, ax_5\}\}$

$\Delta_{\mathcal{T}_1,A3} : \{\{ax_3\}, \{ax_4\}, \{ax_5\}\}$

$\Delta_{\mathcal{T}_1,A6} : \{\{ax_1\}, \{ax_4\}, \{ax_6\}, \{ax_2, ax_3\}, \{ax_2, ax_5\}\}$

$\Delta_{\mathcal{T}_1,A7} : \{\{ax_4\}, \{ax_7\}\}$

Minimal incoherence-preserving sub-TBoxes (MIPS)
MUPS are useful for relating sets of axioms to the unsatisfiability of specific concepts, but they can also be used to calculate MIPS, which relate sets of axioms to the incoherence of a TBox in general (i.e. unsatisfiability of at least one concept in a TBox).

Definition 6 A TBox $\mathcal{T}' \subseteq \mathcal{T}$ is a *minimal incoherence preserving sub-TBox (MIPS)* of $\mathcal{T}$ if, and only if, $\mathcal{T}'$ is incoherent, and every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$ is coherent.

This means that MIPS are minimal subsets of an incoherent TBox preserving unsatisfiability of at least one atomic concept. The set of MIPS for a TBox $\mathcal{T}$ is abbreviated with $mips(\mathcal{T})$. For $\mathcal{T}_1$ we get 3 MIPS: $mips(\mathcal{T}_1) = \{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$

Analogous to MUPS, each element of $mips(\mathcal{T})$ is a minimal conflict set w.r.t. incoherence of TBox $\mathcal{T}$. Hence, from $mips(\mathcal{T})$, a diagnosis for coherence of $\mathcal{T}$ can be calculated, which we denote as $\Delta_{\mathcal{T}}$. From these definitions, we can determine the diagnosis for coherence of $\mathcal{T}_1$:

$\Delta_{\mathcal{T}1} = \{\{ax_1, ax_4\}, \{ax_2, ax_4\}, \{ax_1, ax_3, ax_7\}, \{ax_2, ax_3, ax_7\}, \{ax_1, ax_5, ax_7\}, \{ax_2, ax_5, ax_7\}\}$

The number of MUPS a MIPS is a subset of determines the number of unsatisfiable concepts of which it might be the cause. We will call this number the *MIPS-weight*.

In the example terminology $\mathcal{T}_1$ we found six MUPS and three MIPS. The MIPS $\{ax_1, ax_2\}$ is equivalent to one of the MUPS for A_1, $\{ax_1, ax_2\}$, and a proper subset of a MUPS for A_6, $\{ax_1, ax_2, ax_4, ax_6\}$. Hence, the weight of MIPS $\{ax_1, ax_2\}$ is two. In the same way we can calculate the weights for the other MIPS: the weight of $\{ax_3, ax_4, ax_5\}$ is three, the weight of $\{ax_4, ax_7\}$ is one. Intuitively, this suggests that the combination of the axioms $\{ax_3, ax_4, ax_5\}$ is more relevant than $\{ax_4, ax_7\}$.

Weights are easily calculated, and play an important role in practice to determine relative importance within the set of MIPS, as we experienced in our case-studies which are described in [Schlobach *et al.*, 2007].

2.8.2.3 Pinpoints

Experiments described in [Schlobach, 2005b] indicated that calculating diagnoses from MIPS and MUPS is simple, but computationally expensive, and often impractical for real-world terminologies. For this purpose, we introduced in [Schlobach, 2005a] the notion of a *pinpoint* of an incoherent terminology $\mathcal{T}$, in order to approximate the set of diagnoses. The definition of the set of pinpoints is a procedural one, following a heuristic to ensure that *most* pinpoints will indeed be diagnoses. However, there is no guarantee of minimality, so that not every pinpoint is necessarily a diagnosis.

To define pinpoints we need the notion of a core: MIPS-weights provide an intuition of which combinations of axioms lead to unsatisfiability. Alternatively, one can focus on the occurrence of the individual axioms in MIPS, in order to predict the likelihood that an individual axiom is erroneous. We define cores as sets of axioms occurring in several MIPS. The more MIPS such a core belongs to, the more likely its axioms will be the cause of contradictions.

Definition 7 A non-empty subset of the 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 $\mathcal{T}$.

For our example TBox $\mathcal{T}_1$ we find one 2-ary core, $\{ax_4\}$ of size 1. The other axioms in the MIPS are 1-ary cores. Pinpoints are defined in a structural way.

Definition 8 Let $mips(\mathcal{T})$ be the set of MIPS of $\mathcal{T}$, i.e. a collection of sets of axioms. The set of possible outputs of the following procedure will be called the *set of pinpoints*.

Let $M := mips(\mathcal{T})$ be the collection of MIPS for $\mathcal{T}$, $P = \varnothing$:
(1) Choose in M an arbitrary core $\{ax\}$ of size 1 with maximal arity.
(2) Then, remove from M any MIPS containing $\{ax\}$
(3) $P := P \cup \{ax\}$
Repeat steps (1) to (3) until $M = \varnothing$. The set P is then called a *pinpoint* of the terminology.

As step (1) contains a non-deterministic choice there is no unique *pinpoint* but a set possible of possible outputs of the algorithm: the set of pinpoints.

For our example TBox $\mathcal{T}_1$ with $mips(\mathcal{T}_1)$ = $\{\{ax_1, ax_2\}, \{ax_3, ax_4, ax_5\}, \{ax_4, ax_7\}\}$ we first take the 2-ary core, $\{ax_4\}$. Removing the MIPS containing ax_4 leaves $\{ax_1, ax_2\}$. Hence, there is a non-deterministic choice: if we choose ax_1 to continue $\{ax_4, ax_1\}$ is the calculated pinpoint, otherwise $\{ax_4, ax_2\}$. Both are diagnoses of $\mathcal{T}_1$.

2.8.3 DION: A Bottom-up Approach for Debugging Incoherent Ontologies

In [Schlobach and Huang, 2005, Schlobach and Huang, 2007], an informed bottom-up approach to calculate MUPS with the support of an external DL reasoner is proposed. The

advantage of this approach is that it can deal with any DL-based ontology supported by an external reasoner. Currently there exist several well-known DL reasoners, like RACER[9], FaCT++[10], and Pellet[11], each of which has proved to be reliable and stable. They support various DL-based ontology languages, including OWL-DL.

Given an unsatisfiable concept A and a terminology $\mathcal{T}$ MUPS can be systematically calculated by checking whether A is unsatisfiable in subsets $\mathcal{T}'$ of $\mathcal{T}$ of increasing size. Such a procedure is complete and easy to implement, but infeasible in practice. Even the most simple real-world terminology in our tests in [Schlobach *et al.*, 2007] has an average size of 5 axioms per MUPS and 417 axioms, which requires over 10^{11} calls to the external reasoner.

This observation implies that one has to control the subsets of $\mathcal{T}$ that are checked for satisfiability of A by means of a *selection function*. Such a selection function selects increasingly large subsets which are heuristically chosen to be relevant additions to the currently selected subset. Although this approach is not guaranteed to give us the complete solution set of MUPS it provides an efficient approach for debugging inconsistent terminologies. We will now formally introduce the core notions of selection functions and relevance.

In [Huang and van Harmelen, 2006] two different selection functions are defined. The most simple one is based on co-occurrence of concept names in axioms. In this approach, they focus on unfoldable TBoxes,[12] they use a slightly more complex selection function here. The basic idea is that an axiom ax is relevant to a concept name A if, and only if, A occurs on the left-hand side of ax. In a way this variant of the bottom-up approach mimics the unfolding procedure in order to restrict the number of tests needed. This is also the one implemented in the DION system.

Using the concept relevance relation, we can define a particular selection function. For a terminology $\mathcal{T}$ and a concept A, define a selection function s as follows:

Definition 9 The *concept-relevance based selection function* for a TBox $\mathcal{T}$ and a concept A is defined as

(i) $s(\mathcal{T}, A, 0) = \varnothing$;
(ii) $s(\mathcal{T}, A, 1) = \{ax \mid ax \in \mathcal{T}$ and ax is concept-relevant to $A\}$;
(iii) $s(\mathcal{T}, A, k) = \{ax \mid ax \in \mathcal{T}$ and ax is concept-relevant to an element in $s(\mathcal{T}, A, k-1)\}$ for $k > 1$.

An informed bottom-up approach to obtain MUPS is proposed in [Schlobach and Huang, 2007]. In logics and computer science, an increment-reduction strategy is often used to find minimal inconsistent sets [de la Banda *et al.*, 2003]. Under

[9] http://www.sts.tu-harburg.de/~r.f.moeller/racer/
[10] http://owl.man.ac.uk/factplusplus/
[11] http://www.mindswap.org/2003/pellet/
[12] Remember that the top-down is defined for unfoldable TBoxes only.

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

Figure 2.4: **MUPS_bottomup**($\mathcal{T}, A$)

this approach, the algorithm first finds a collection of inconsistent subsets of an inconsistent set, before it removes redundant axioms from these subsets. Similarly, a heuristic procedure for finding MUPS of a TBox $\mathcal{T}$ and an unsatisfiable concept-name A consists of the following three stages:

- **Expansion** : Use a relevance-based selection function to find two subsets Σ and S of $\mathcal{T}$ such that a concept A is satisfiable in S and unsatisfiable in $S \cup \Sigma$.
- **Increment**: Enumerate subsets of Σ to obtain the sets S" such that the concept A is unsatisfiable in S" $\cup\, S$. We call those sets A-*unsatisfiable sets*.
- **Reduction**: Remove redundant axioms from those A-unsatisfiable sets to get MUPS.

Figure 2.4 describes an algorithm **MUPS_bottomup**($\mathcal{T}, A$) based on this strategy to calculate MUPS. The algorithm first finds two subsets Σ and S of $\mathcal{T}$ by increasing the relevance degree k on the selection function until A is unsatisfiable in $S \cup \Sigma$ but satisfiable in Σ. Compared with $\mathcal{T}$, the set Σ can be expected to be relatively small. The algorithm then builds the power-set of Σ to get A-unsatisfiable sets by adding an axiom $ax \in \Sigma$ in each iteration of the loop to the sets S' in the working set W. If A is satisfiable in

Figure 2.5: **MinimalityChecking(M($\mathcal{T}, A$))**

for all $M \in M(\mathcal{T}, A)$ **do**
  $M' := M$
  for all $ax \in M'$ **do**
    if A unsatisfiable in $M' - \{ax\}$ **then**
      $M' := M' - \{ax\}$
    end if
  end for
  $M(\mathcal{T}, A) := M(\mathcal{T}, A) - \{M\} \cup \{M'\}$
end for
return $M(\mathcal{T}, A)$

$S' \cup \{ax\}$, then the set $S' \cup \{ax\}$ is added to the working set to build up the union of each elements of the power-set of Σ with the set S.[13] If A is unsatisfiable in $S' \cup \{ax\}$, then add the set $S' \cup \{ax\}$ into the resulting set $M(\mathcal{T}, A)$ instead of the working set W. This avoids the calculation of the full power-set of Σ because any superset of $S' \cup \{ax\}$ in which A is unsatisfiable is pruned. Finally, by checking minimality we obtains MUPS. The procedure to check minimality of the calculated subsets of $\mathcal{T}$ is described in Figure 2.5.

2.8.4 RepairTab: A Heuristic Approach for Repairing Unsatisiable Ontologies

[Lam, 2007, Lam *et al.*, 2006] investigate the heuristics used by ontology engineers to resolve inconsistencies in ontologies. An empirical study has been conducted to acquire a variety of such heuristics from ontology engineers. Some of the acquired heuristics are already incorporated in existing tools, however, a sizeable number of additional useful heuristics have also discovered. Moreover, the usefulness of incorporating these heuristics in a software tool is investiagted to guide non-expert ontology modelers to select appropriate axioms for modification. The result of the usability study shows that the heuristics are useful to help ontology users debug ontologies. This work shows that the approach of acquiring human heuristics and encoding them in a tool is viable for ontology debugging.

The heuristics used by ontology engineers have been acquired to resolve inconsistencies in ontologies. These heuristics have been formalized to evaluate confidence in axioms, so as to suggest to the user which axioms should be removed or modified. This approach is implemented in the RepairTab system, which is a plug-in of Protege 3.3.

Given an ontology with unsatisfiable concepts, we use the tableau-tracing technique to pinpoint the sets of axioms which cause the concepts to be unsatisfiable. At least one

[13] Namely $\{S'/S | S' \in W\} \subseteq \mathcal{P}(\Sigma)$

axiom from each set of axioms must be removed/modified to resolve an unsatisfiability; however we need to identify which axiom should be removed/modified. To do this we assign *confidence* values to axioms. If an axiom is assigned a high confidence value we recommend that the axiom should be preserved. If an axiom is assigned a low confidence value this means that we recommend that the axiom should be removed or modified. Confidence values are assigned by a set of heuristics; an example is the "impact" heuristic which assigns high confidence to axioms whose removal would impact many of the named concepts in the ontology (the aim is to minimise the loss of information from the ontology).

The confidence function below provides a ranking over the axioms, allowing us to indicate axioms recommended for modification or preservation. As we have to compare the relative confidence values of axioms, the heuristic formulas below transform the computed values into the range $[-1, 1]$ (normalisation). Axioms are initially presumed to have 0 confidence; axioms with values close to $+1$ are strongly recommended for preservation, while those with values close to -1 are strongly recommended for removal. Axioms for which a confidence value is not determined for a particular heuristic are assigned the default confidence value 0.

Definition 10 (Confidence) Let α be an axiom in an ontology $\mathcal{O}$, the confidence of α is a function

$$confidence : \alpha \rightarrow [-1, 1]$$

We have acquired a variety of heuristics from our empirical study. We formalise the heuristics which allow us to evaluate the confidence of axioms further. We combine our heuristics with those already incorporated in existing debugging tools (i.e., arity and number of lost entailments) and divide them into the following categories:

1. Impact of removal on the ontology – two heuristics are presented to analyse the loss of entitlements due to removing axioms

2. Knowledge of ontology structure – five novel heuristics are proposed to analyse

   (a) disjointness between siblings,

   (b) disjointness between non-siblings,

   (c) sibling patterns,

   (d) length of paths in a concept hierarchy, and

   (e) depth of concepts in a concept hierarchy

3. Linguistic heuristic – one heuristic is presented to analyse the similarities of the names of the concepts

2.8.5 RaDON: A System for Reasoning and Diagnosis in Ontology Networks

RaDON[14] is a system for Reasoning and Diagnosis in Ontology Networks, which is developed by the AIFB. RaDON - Repair and Diagnosis for Ontology Networks - provides a set of techniques for dealing with inconsistencies and incoherence in ontologies. In particular, RaDON supports various strategies and consistency models for distributed and networked environments.

RaDON is a system that extends the capabilities of existing reasoners with functionalities to deal with inconsistencies. These additional functionalities are made accessible via extensions to the DIG interface. The idea of extending the DIG interface with non-standard reasoning services has been developed originally in the SEKT project and has for example been applied in PION and evOWLution.

These functionalities are accessible in two different ways in the RaDON system and the KAON2 OWL Tools: While the implementation of RaDON is based on extensions to the DIG interface, the KAON2 OWL Tools allow a command line based interface to the functionalities.

The implemented functionalities basically support the individual steps of the RaDON approach to resolving inconsistency and incoherence. In particular, the implementation supports the following tasks of the process:

- **Consistency checking of TBox and ABox**: Consistency checking is a standard reasoning task provided by DL reasoners. We simply perform separate checks for the TBox, ABox, and their union ;
- **Checking the Coherence of TBox**: By definition, we simply need to check the coherence of all concepts of the ontology. This again is a standard reasoning task;
- **Consistency checking of TBox and ABox**. For each unsatisfiable concept, its minimal unsatisfiability-preserving sub-TBox (MUPS) will be given. In such case, we will point out all the minimal incoherence-preserving sub-TBox (MIPS) where the incoherence occurs.

2.8.6 Other Approaches for Debugging

The MINDSWAP group at the University of Maryland has done significant work in this area, culminating in the recent thesis of Kalyanpur [Kalyanpur, 2006]. The work investigates two different approaches, one based on modifying the internals of a DL reasoner (the so-called “glass box” approach), and one based on using an unmodified external reasoner (the “black box” approach).

[14] http://radon.ontoware.org/

The glass box approach is closely related to our top-down approach, and is based on the techniques in [Baader and Hollunder, 1995]. The work deals with OWL-Lite, except for max cardinality roles, and is efficient since it avoids having to do full tableau saturation (details are in [Kalyanpur *et al.*, 2005]). The work in [Baader and Hollunder, 1995] is particularly noteworthy: Although that paper is about a different topic (computing extensions for a certain class of default Description Logics), it turns out that one of the algorithms is very similar to the top-down approach. The main difference to Baader et. al's work is that they consider ABoxes instead of TBoxes, and the purpose of the algorithm (computing default extensions vs. computing diagnoses).

The black box approach (i.e.. detecting inconsistencies by calling an unmodified external DL reasoner) is based on Reiter's Hitting Set algorithm (similar to our work in [Schlobach, 2005b]), and also closely related to a proposal of Friedrich et al. [Friedrich and Shchekotykhin, 2005] who bring the general diagnostic theories from [Reiter, 1987] to bear for diagnosing ontologies. An interesting difference with the work reported in this document is that Friedrich et. al use generic diagnoses software. As we do in our bottom-up method, they use a DL reasoner as oracle.

Kalyanpur also proposes a method for "axiom pinpointing"[15], which rewrites axioms into smaller ones, and then debugs the resulting ontology after rewriting, with the effect that a more precise diagnosis is obtained. Early results have been reported in [Kalyanpur *et al.*, 2006]

A second pinpointing technique called "error pinpointing" by Kalyanpur is similar to what we call pinpointing here. Interestingly, Kalyanpur has performed user studies which reveal that a combination of axiom pinpointing (i.e.. breaking large axioms up into smaller ones) and error pinpointing (ie. finding the errors which lie at the root of a cascading chain of errors) together seems to be the cognitively most efficient support for users.

Finally, a significant extension to the work in [Schlobach and Cornet, 2003b] was published in [Meyer *et al.*, 2006], where the authors extend the saturation based tableau calculus with blocking conditions, so that general TBoxes can be handled.

We have reported the benchmarking experiments on the debugging approaches in [Schlobach *et al.*, 2007]. In Chapter 4 of this document we will report the benchmarking experiments on the comparison between the approaches of reasoning with inconsistent ontologies and the approaches of debugging inconsistent ontologies.

[15] A different use of the word pinpointing from our use, and even from the identical term "axiom pinpointing" in [Schlobach and Cornet, 2003b].

Chapter 3

A Framework for Benchmarking of Processing Inconsistent Ontologies

3.1 Framework

3.1.1 Measuring the Quality of Query Answers

In ontology engineering, evaluation and benchmarking target at software products, tools, services, and processes. Those objects are called tested systems. Evaluation and benchmarking are the systematic determination of merit, worth, and significance of tested systems. Those merit, worth, and significance are characterized as a value relation, which is considered as a preference relation, i.e., a partial order set $\langle A, \succ \rangle$.

The answer value set for reasoning with inconsistent ontologies usually consists of three values $accepted, rejected$, and $undetermined$ as discussed in the previous chapter. We will develop gold standards which represent intuitive answers from a human for queries on the system of processing inconsistent ontologies. Thus, we can compare the answers from the tested system/approach with the gold standard which is supposed to be intuitively true by a human to see to what quality of query answers provided by tested systems.

For a query, there might exist the following difference between an answer from the tested system/approach and its intuitive answer in a gold standard.

- **Intended Answer**: the system's answer is the same as the intuitive answer;
- **Counter-intuitive Answer**: the system's answer is opposite to the intuitive answer. Namely, the intuitive answer is 'accepted' whereas the system's answer is 'rejected', or vice versa.
- **Cautious Answer**: The intuitive answer is 'accepted' or 'rejected', but the system's answer is 'undetermined'.

- **Reckless Answer**: The system's answer is 'accepted' or 'rejected' whereas the intuitive answer is 'undetermined'. We call it a reckless answer, because under this situation the system returns just one of the possible answers without seeking other possibly opposite answers, which may lead to 'undetermined'.

Therefore, a value set

$$\{intended\_answer, cautious\_answer, reckless\_answer, counter\_intuitive\_answer\},$$

can be introduced for the evaluation of answers with golden standards. An intended answer is considered as a best one, whereas a counter intuitive answer is considered as a worst one. Cautious answers are usually not considered as wrong answers, whereas reckless answers may give wrong answers. Thus, a preference relation on the value set can be like this:

$$\{intended\_answer \succ cautious\_answer,$$
$$cautious\_answer \succ reckless\_answer,$$
$$reckless\_answer \succ counter\_intuitive\_answer\}$$

Based on this preference order, we can measure the quality of query answers by the following answer rates:

- **IA Rate**, which counts only intended answers. Namely the Intended Answer Rate is defined as the ratio of the amount of Intended Answers to the total amount of the answers.
- **IC Rate**, which counts non-error answers. Namely, IC Rate = (Intended Answers +Cautious Answers)/TotalAnswerNumber.

3.1.2 Basic Definitions

In order to clarify our ideas about the framework for benchmarking of processing inconsistent ontologies, we introduce a set of relevant definitions in the following

- **Tested systems**: Tested systems are ones which are targeted by the objectives of evaluation or benchmarking. A tested system can be characterized as an input-output function, alternatively, called a characteristic function of the tested system. Namely, it maps a tuple of the input parameters into an output value.
- **Value relation**: A value relation is a preference relation on a set of values. Namely, a value relation is characterized as a partial order set.
- **Evaluation**: Evaluation is the systematic determination of the values of tested systems with respect to its partial ordered value relation.

- **Benchmarking**: Benchmarking is a continuous process for improving by systematically evaluating tested systems, and comparing them to those considered to be the best. Namely, benchmarking is a continuous processing of evaluation.

- **Golden Standard**: A golden standard is an evaluation function which maps queries into answers with confidence values[1].

3.1.3 Workflows of Evaluation and Benchmarking

Common data sets and common golden standards are usually used for an evaluation of different tested systems. Those systems may be heterogeneous with respect to their input data. For example, a reasoner may support only OWL data, whereas another reasoner may support only DIG data. That needs a data translator to convert data sets represented in a standard format into the data sets which are represented in a format which is supported by a tested system. Based on comparison between test results and golden standards, result evaluation can be done manually, semi-automatically, or automatically. The output of the result evaluation and the implication are further analyzed by an evaluation analysis. The methods of statistics and visualization are usually introduced in the phase for better illustration. The evaluation results will be ranked with respect to its value relation. Finally, it leads to an evaluation report which concludes the values of tested systems and explain the reasons why the system behave differently. An investigate is usually made to detect the problem of tested systems based on the analysis of the evaluation. The workflow of evaluation is shown in Figure 3.1.

As discussed above, benchmarking is a continuous processing of evaluation. Therefore, for benchmarking, evaluation results are used further for the improvement of tested systems. That would usually lead to new versions of tested systems. Based on a benchmarking analysis, new test data sets may be re-designed or previous data sets are adjusted for further evaluation with respect to some targeted problems. The workflow of benchmarking is shown in Figure 3.2. In this benchmarking activity, we have instantiated the Knowledge Web benchmarking methodology[Garcá-Castro *et al.*, 2005] to the particular case of benchmarking the processing of inconsistent ontologies.

3.1.4 Taxonomy

There are various perspectives on tested systems for evaluation and benchmarking. We distinguish the following variants and dimensions:

- Black box vs. Glass box: Tested systems (products/services/processes) can be considered as black boxes or glass boxes. The former means that the internal details of the tested systems are not transparent to their evaluators.

[1] We will discuss gold standards in Section 3.2.

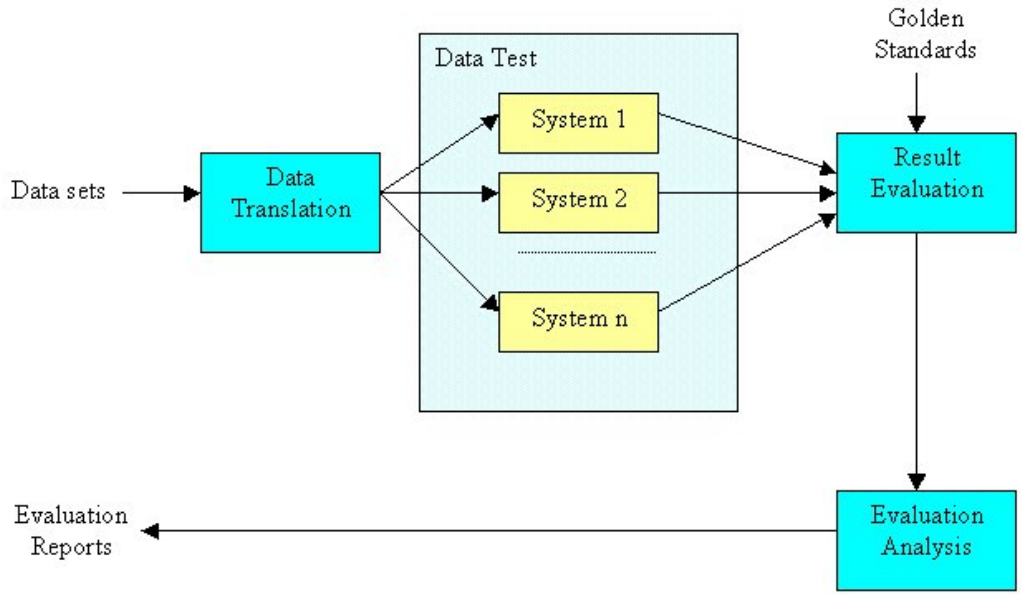

Figure 3.1: Workflow of evaluation.

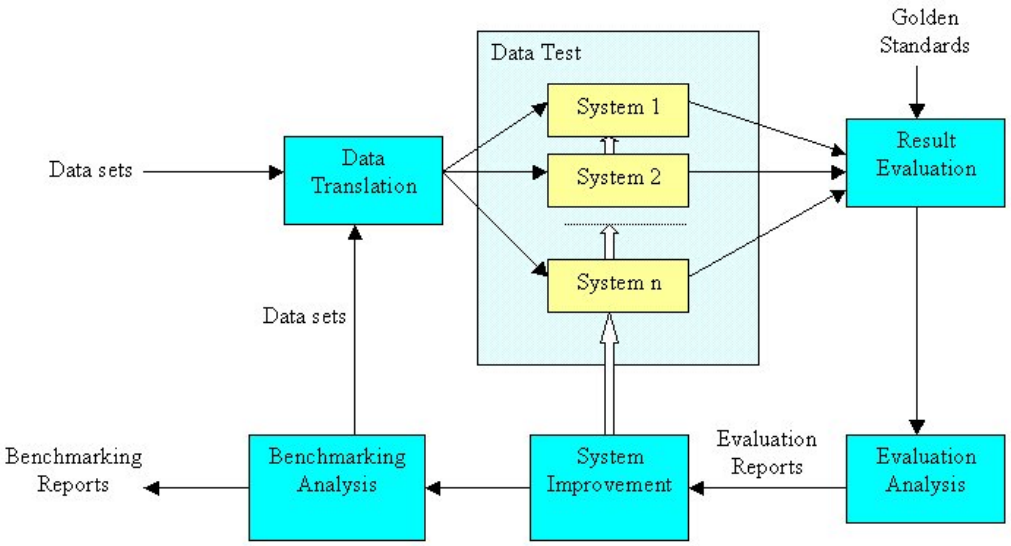

Figure 3.2: Workflow of benchmarking

- Internal comparison vs. External comparison: the former refers to the comparison of various functions inside a tested system, whereas the latter refers to the comparison among different systems.

- Qualitative evaluation vs. Quantitative evaluation. The latter refers one in which the value relation is a subset of real numbers. The former refers to one in which the value relation is not a total order set. A Boolean evaluation is one in which the value relation is a Boolean set $\{0, 1\}$ (or $\{unintended, intended\}$, or $\{good, bad\}$).

In the benchmarking of reasoning with inconsistent ontologies, we will conduct an internal comparison of the system. Namely we will evaluate and compare various selection functions of the system and various approaches of extension strategies and various strategies of over-determined processing. The result of the benchmarking is expected to provide a quantitative results. We will suggest possible improvements on the functionalities and implementation of the tested system. Thus, we can considered the tested system as a Glass box.

3.2 A Specification Language for Golden Standards

Manual evaluation and analysis of test results are usually time consuming, labor intensive, and error prone. The formalism of golden standard will pave a way for automatic or semi-automatic evaluation and analysis of test results.

A golden standard is an evaluation function which maps queries into answers with confidence values. For reasoner benchmarking, a gold standard is a (partial) function which maps queries into (intuitive) answers with a confidence value. For example, for benchmarking inconsistency processing, we considered the answer set $\{accepted, rejected, undetermined\}$. For a query "are birds animals?", the expected answer is intuitively considred as "accepted" with confidence value "1.0". However, for the query "are men animals?", the expected answers may be well suitable to be specified as an answer with lower confidence value, say, "accepted" with confidence value "0.4", "rejected" with confidence value "0.4", and "undetermined with the confidence value "0.2". Namely, we use the confidence values to represent some kinds of uncertainty of expected answers. The confidence values can be obtained by various approaches, like from questionnaires, statistics, machine learning, etc.

We design gold standards which are independent from a specific ontology. Namely, it is up to evaluators/users to decide which ontologies can be applied with respect to a golden standard.

In the following, we develop a golden standard specification language in which the concept language is based on the DIG data format. Thus, it is an XML file, which is easy to be used and read. Of course an alternative is to develop the golden standard specification language by using OWL.

The following is an example of a golden standard:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<goldenStandard xmlns="http://wasp.cs.vu.nl/knowledgeweb/d2163/lang"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://wasp.cs.vu.nl/knowledgeweb/d2163/gd.xsd">

  <name value="Kweb golden standard example 1" version="0.0.1"/>
  <comment text="just an example, which is independent from any
    particular ontology. It is up to evaluators to decide
    which ontology can be applied"/>

      <query id="Are birds animals?" querytype="subsumes">
        <subsumes>
          <catom name="#Animal"/>
          <catom name="#bird"/>
         </subsumes>
        <expectedAnswers>
          <answer value="accepted" confidence="1"/>
        </expectedAnswers>
      </query>

      <query id="Are men animals?" querytype="subsumes">
        <subsumes>
          <catom name="#Animal"/>
          <catom name="#man"/>
        </subsumes>
        <expectedAnswers>
          <answer value="accepted" confidence="0.4"/>
          <answer value="undetermined" confidence="0.2"/>
          <answer value="rejected" confidence="0.4"/>
          <comment text="just an example which shows the possibility
                of multiple answers in a golden standard"/>
        </expectedAnswers>
     </query>

</goldenStandard>
```

Which specifies the name and the version of the golden standard. Each query consists of a detailed query statement (in the DIG query language) and its expected answer specification. Each expected answer is attached by a confidence value.

3.3 A Benchmarking Suite for Processing Inconsistent Ontologies

[Garcá-Castro *et al.*, 2005] suggests that desirable properties for a benchmark suite should be: Accessibility, Affordability, Representativity[2], Portability[3], Scalability[4], Robustness[5], and Consensus[6]. The design of the benchmarking suite for processing inconsistent ontologies considers those properties to some extent. In particular, the benchmarking suite is designed to be one is accessible for any user who may have no any knowledge of programming for the accessbility. It is affordable, because it is designed to be use freewares such as SWI prolog. It is portable because it can be used on different platforms.

We have implemented a benchmarking suite for processing inconsistant ontologies. The benchmarking suite is implemented by using SWI prolog[7], and powered by the XDIG, an extended DIG interface for Prolog[Huang and Visser, 2004, Wielemaker *et al.*, to appear]. However, the users of the benchmarking suite need no any knowledge of Prolog to use it.

3.3.1 Benchmarking Tools

The benchmarking suite consists of the following benchmarking tools:

- Gold Standard Authoring Tool. As its name implies, it is a tool for gold standard authoring. It provides various approaches for the authoring, which can create a gold standard from a scratch, or extend the current gold standard with newly added data.
- Query Test Tool. It is a program which is used to create queries, post queries to tested systems, and record the answers from tested systems automatically.
- Evaluation Tool. It is a tool which is designed for compare the answers from the tested system with a gold standard automatically.

The relation among the benchmarking tools is shown in Figure 3.3.

[2]The actions that perform the benchmarks that compose the benchmark suite must be representative of the actions that are usually performed on the system.

[3]The benchmark suite should be executed on as wide a variety of environments as possible, and should be applicable to as much systems as possible.

[4]The benchmark suite should be parameterised to allow to scale the benchmarks with varying input rates. It should also scale to work with tools or techniques at different levels of maturity. It should be applicable to research prototypes and commercial products.

[5]The benchmark suite must consider unpredictable environment behaviours, and should not be sensitive to factors not relevant to the study.

[6]The benchmark suite must be developed by experts that provide their knowledge about the domain, and are able to identify the key problems. It should also be assessed and agreed on by the whole community.

[7]http://www.swi-prolog.org

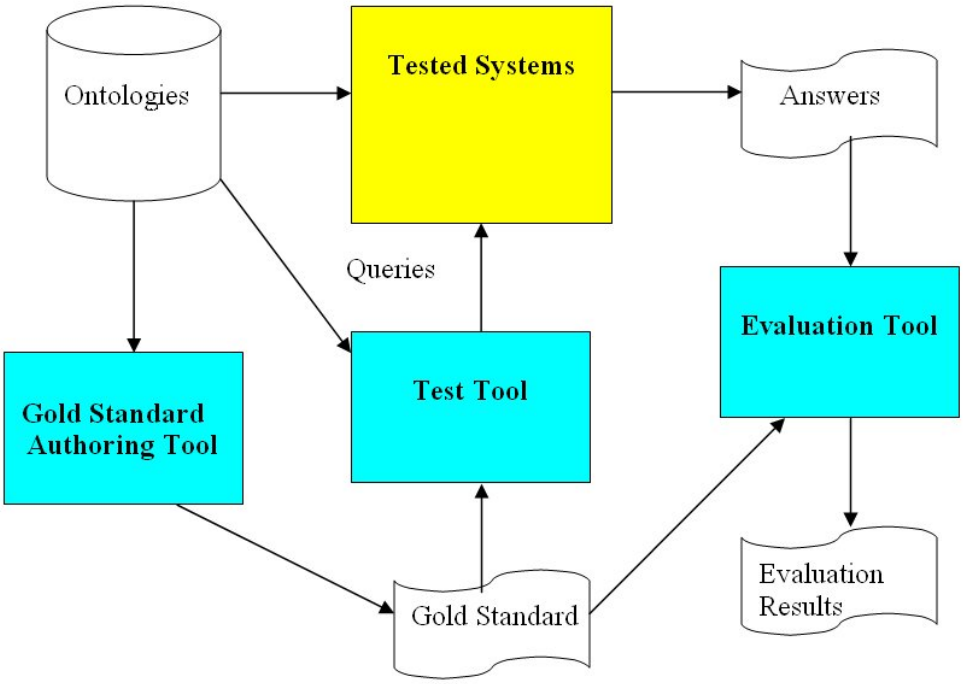

Figure 3.3: Architecture of Benchmarking Suite

In the following we will provide a detailed manual how the benchmarking tool can be installed and used. First of all, you can download the benchmarking suite from the VUA KnowledgeWeb website:

$$http://wasp.cs.vu.nl/knowledgeweb/d2163.$$

Unzip the suite into a directory and make sure that SWI Prolog has been installed in your computer.

3.3.2 Gold Standard Authoring Tool

The Gold Standard Authoring Tool can be launched by double clicking on the file 'gstool.pl'. If you encounter the error 'out of global stack' because of too large data in the gold standard, you can launch the Gold Standard Authoring tool by double clicking on the file 'gstool_bigglobalStack.bat' after changing the path setting "C:/Program Files/pl/bin/plwin.exe" in the bat file if SWI-prolog is installed in other directories. The Gold Standard Authoring Tool will display a window, in which you can check the setting such as the gold standard file name, the working directory, etc, and change them accordingly, as shown in Figure 3.4. The selected conceptpair file name allows the system to load a list of pre-selected concept pairs for queries. User can also different strategies to

Figure 3.4: The Setting Interface of the Gold Standard Authoring Tool

create a set of subsumption queries. The current verison of the gold standard authoring tool supports the strategy 'random' only. Namely, the system selects concept pairs for queries randomly.

3.3.3 Test Tool

The Test Tool is a program for automatically posting queries into a tested system and record the responses from the system. This tool is used in the command line of SWI-Prolog. However, users are not required to have a knowledge of Prolog.

Users can do the following steps to use the test tool.

1) Use a text editor to edit the file testtool/testtool.pl on the following setting lines:

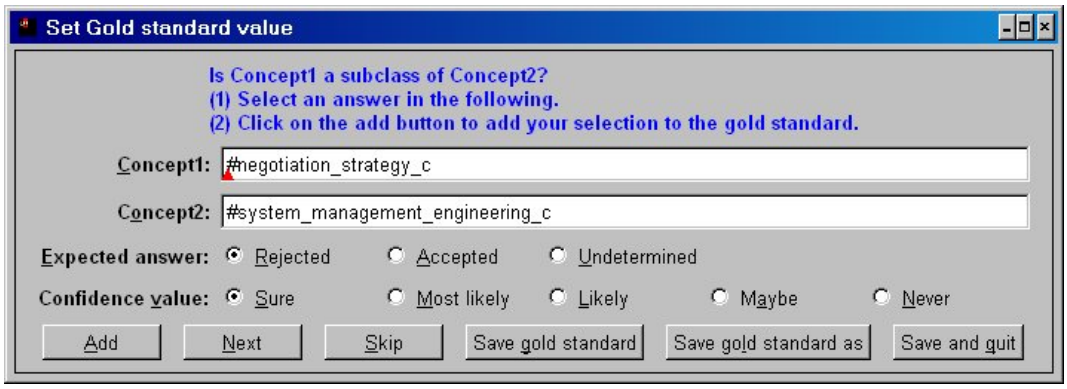

Figure 3.5: The Interface for Adding Gold Standard Data

```
testtool_setting(url, [protocol(http), host(localhost),
         port(8001), path(/)]).
testtool_setting(ontology,   'proton_50_rudis.dig.xml').
testtool_setting(log_type, load_and_ask_log).
testtool_setting(test_type, subsumption_list).
testtool_setting(selected_concept_list, L):-
   L=['Airport', 'Book', ...].
```

If the port number of the tested system is not '8001', then change it accordingly. Before the test, copy the tested ontology into the same directory where the program testtool.pl locates. Change the tested ontology file name in the test tool setting line.

The log type setting provides several modes for loading tested ontologies and recording the results:

- log_type=load_and_ask_log: load the tested ontologies, create the corresponding queries, post the queries, and log the queries in the directory 'ask'.
- log_type= load_and_no_ask_log: load the tested ontologies, create the corresponding queries, and post the queries without the log of the queries in the directory 'ask'.
- log_type=no_load_and_no_ask_log: obtain the list of all concepts from the tested system in which an ontology has been loaded, and create the corresponding queries, post the queries without the log of the queries.

The test type provides several modes for tests:

- test_type=subsumption_full: create the subsumption queries for all concept pairs in the tested ontology.
- test_type=subsumption_list: create the subsumption queries for a selected concept list which is defined in the selected_concept_list of the testtool setting.

The list L in the following setting line

```
testtool_setting(selected_concept_list, L):-
  L=['Airport', 'Book', ...].
```

defines the selected concept list, from which the test tool can create corresponding queries based on the concept pairs from this pre-selected concept list automatically.

After the change of the setting, users can launch the file 'testtool.pl' to start the test. The responses from the tested system are recorded in the directory 'response' automatically. Namely, users can follow the following procedure to do a test:

- Changing the setting. Use any text editor to check and change the setting in the file 'testtool.pl',
- Starting the tested system. Start a tested system like the PION system,
- Launching the testtool until the test is done,
- Checking the test results. Use any text/xml viewers to see the results which are stored in the directory 'response'.

3.3.4 Evaluation Tool

The Evaluation tool is used to compare the response data from tested systems with a gold standard. The corresponding queries in the response files are identified by their query ID.

Before starting the evaluation tool, users can check and change the following setting lines in the file 'evaluationtool.pl':

```
default_value(goldstandard_file, 'gs.xml').
default_value(dataDirectory, './response').
default_value(evaluation_result_file, 'evaluation_result.xml').
```

The setting above states that the file of the gold standard is 'gs.xml', the directory of the response data is './response', and the comparison result will be recorded in the file 'evaluation_result.xml'.

The user can also change the evaluation function in the following lines in the file 'evalutiontool.pl':

```
gs_comparison_value(accepted,rejected, counterintuitive):-!.
gs_comparison_value(rejected,accepted, counterintuitive):-!.
```

```
- <evaluation>
  - <result>
    - <result>
        <query>does Airport subsume Lake?</query>
        <answer>rejected</answer>
        <expectedAnswer>rejected</expectedAnswer>
        <confidence>1</confidence>
        <conclusion>intended</conclusion>
        <intended>1</intended>
        <step>3</step>
        <odpstep>0</odpstep>
      </result>
    - <result>
        <query>does Airport subsume Location?</query>
        <answer>undetermined</answer>
        <expectedAnswer>undetermined</expectedAnswer>
        <confidence>1</confidence>
        <conclusion>intended</conclusion>
        <intended>1</intended>
        <step>4</step>
        <odpstep>1</odpstep>
      </result>
    - <result>
        <query>does Airport subsume MagazineIssue?</query>
        <answer>rejected</answer>
        <expectedAnswer>rejected</expectedAnswer>
        <confidence>1</confidence>
        <conclusion>intended</conclusion>
        <intended>1</intended>
```

Figure 3.6: Evaluation_result.xml

esult	query	answer	expectedAnswer	confidence	conclusion	intended	step	odpstep	cautious	reckless	counterintuitive
	does Airport subsume Airport?	accepted	accepted	1	intended	1	2	0			
	does Airport subsume Book?	rejected	rejected	1	intended	1	3	0			
	does Airport subsume Canal?	undetermined	rejected	1	cautious		4	106	1		
	does Airport subsume CEO?	undetermined	rejected	1	cautious		5	44	1		
	does Airport subsume Coast?	undetermined	rejected	1	cautious		4	1	1		
	does Airport subsume Document?	rejected	rejected	1	intended	1	3	0			
	does Airport subsume Harbor?	rejected	rejected	1	intended	1	3	19			
	does Airport subsume HydrographicStruc	undetermined	rejected	1	cautious		3	1	1		
	does Airport subsume Island?	undetermined	rejected	1	cautious		4	1	1		
	does Airport subsume Lake?	rejected	rejected	1	intended	1	3	0			
	does Airport subsume Location?	undetermined	undetermined	1	intended	1	4	1			
	does Airport subsume MagazineIssue?	rejected	rejected	1	intended	1	4	1			
	does Airport subsume Manager?	undetermined	rejected	1	cautious		3	14	1		
	does Airport subsume Market?	undetermined	rejected	1	cautious		4	1	1		
	does Airport subsume Object?	undetermined	rejected	1	cautious		4	1	1		
	does Airport subsume OlympicGames?	rejected	rejected	1	intended	1	3	1			
	does Airport subsume Product?	undetermined	undetermined	1	intended	1	5	1			
	does Airport subsume Reservoir?	undetermined	rejected	1	cautious		4	376	1		
	does Airport subsume Sea?	rejected	rejected	1	intended	1	3	0			
	does Airport subsume Statement?	undetermined	rejected	1	cautious		4	1	1		
	does Airport subsume TransportFacility?	undetermined	undetermined	1	intended	1	4	1			
	does Airport subsume Valley?	undetermined	rejected	1	cautious		4	1	1		
	does Airport subsume WaterRegion?	rejected	rejected	1	intended	1	3	1			
	does Book subsume Airport?	rejected	rejected	1	intended	1	3	0			
	does Book subsume Book?	accepted	accepted	1	intended	1	2	0			
	does Book subsume Canal?	undetermined	rejected	1	cautious		4	1	1		
	does Book subsume CEO?	undetermined	rejected	1	cautious		4	1	1		
	does Book subsume Coast?	rejected	rejected	1	intended	1	4	0			

Figure 3.7: Evaluation_result.xls

```
gs_comparison_value(accepted,undetermined, reckless):-!.
gs_comparison_value(rejected,undetermined, reckless):-!.
gs_comparison_value(undetermined,accepted, cautious):-!.
gs_comparison_value(undetermined,rejected, cautious):-!.

gs_comparison_value(A,A, intended):-!.

gs_comparison_value(_,_, unknown).
```

The first, the second, and the third parameters in the predicate gs_comparison_value correspond the query answer from the response data, the expected answer from the gold standard, and the evaluation result respectively. Therefore, the lines above define the evaluation function which is described in Section 3.1.1.

Users can define their own evaluation functions by changing the definition of the predicate 'gs_comparison_value' in the file 'evaluationtool.pl'. For example, we can use it to define an evaluation function which intrepretes boolean answers (i.e, 'true' or 'false') into multi-valued answers for an evaluation of the answer quality by a debugging system, which is used in the next chapter.

For the interpretation in which 'true' is interpreted as 'accepted' and 'false' as 'rejected' , we define the evaluation funciton in the Evaluation Tool 'evaluationtool.pl' as follows:

```
gs_comparison_value(false,accepted, counterintuitive):-!.
```

```
gs_comparison_value(true,accepted, intended):-!.

gs_comparison_value(true,rejected, counterintuitive):-!.
gs_comparison_value(false,rejected, intended):-!.

gs_comparison_value(true,undetermined, reckless):-!.
gs_comparison_value(false,undetermined, reckless):-!.
```

For the interpretation in which 'true' is intrepreted as 'accepted, and 'false' as 'rejected' or 'undetermined', we define the evaluation funciton in the Evaluation Tool 'evaluationtool.pl' as follows:

```
gs_comparison_value(false,accepted, cautious):-!.
gs_comparison_value(true,accepted, intended):-!.

gs_comparison_value(true,rejected, counterintuitive):-!.
gs_comparison_value(false,rejected, intended):-!.

gs_comparison_value(true,undetermined, reckless):-!.
gs_comparison_value(false,undetermined, intended):-!.
```

The comparison results are recorded in the file 'evaluation_result.xml', as shown in Figure 3.6.

This xml file of the evaluation results can be converted into an excel file by using the program swift_x2e, or any other software which supports the same functionality. The resulting excel file can be used to obtain the statistic data of the benchmarking, as shown in Figure 3.7.

3.4 Data Sets

We are collecting the following data sets of inconsistent ontologies for the benchmarking.

- **Inconsistent Mapping**: Inconsistent ontologies are created by joining paris of ontologies using automatically generated mappings. There are 36 data sets than contain inconsistencies. Contributors: Christian Meilicke and Heiner Stuckenschmidt (KR&KM Research Group of Manheim University)
- **Inconsistent Learning**: Inconsistent ontologies are created by human annotation of disjointness on ontology learning data. There are 9 data sets which contain inconsistencies. Contributor: Johanna Völker (University of Karlsruhe)

- **Inconsistency Processing of SEKT WP3**: Inconsistent ontologies are created in the SEKT WP3.4 (Reasoning with Inconsistent Ontologies) and SEKT WP3.6 (Diagnosis and repair of Inconsistent Ontologies). There are 7 data sets which contain inconsistencies. Contributors: Zhisheng Huang and Stefan Schlobach (Vrije University Amsterdam).

3.4.1 Inconsistent Mapping

The inconsistent ontologies available at this data set are created by joining pairs of ontologies using automatically generated mappings. The concepts of the original ontologies are connected inside the resulting ontologies by adding additional (and sometimes incorrect) axioms. These axioms result from a translation of the matching-results of the Ontology Alignment Evaluation Initiative 2006.

The results of the matching systems Falcon-AO, COMA++, HMatch and Owl-CtxMatch have been chosen to create the "union" of two ontologies. Therefore, the equivalence correspondences created by those matching systems have been translated into a pair of corresponding subsumptions axioms. Since those matching systems often produce erroneous mappings the resulting ontologies contain in many cases unsatisfiable classes. The following tables give an overview about the size of the resulting ontologies and the number of unsatisfiable concepts. Three matching systems generated symmetric mappings (more precise, the same result for the input ontologies $\langle Source, Target\rangle$ and $\langle Target, Source\rangle$), while HMatch produces different results with respect to source and target ontology being exchanged.

All resulting ontologies as well as the manually evaluated mappings can be downloaded from the website:

`http://webrum.uni-mannheim.de/math/lski/ontdebug/index.html#download`

Falcon-AO

Number of unsatisfiable classes / all classes

	CRS	PCS	CMT	CONFTOOL	SIGKDD	EKAW
CRS	-	0 / 38	0 / 44	2 / 53	2 / 64	0 / 87
PCS		-	0 / 53	0 / 62	0 / 73	0 / 97
CMT			-	0 / 68	0 / 79	0 / 103
CONFTOOL			-	0 / 88	7 / 112	
SIGKDD				-	0 / 123	
EKAW					-	

COMA++

Number of unsatisfiable classes / all classes

	CRS	PCS	CMT	CONFTOOL	SIGKDD	EKAW
CRS	-	2 / 38	0 / 44	2 / 53	0 / 64	7 / 87
PCS		-	0 / 53	8 / 62	0 / 73	6 / 97
CMT			-	10 / 68	0 / 79	14 / 103
CONFTOOL				-	13 / 88	5 / 112
SIGKDD					-	0 / 123
EKAW						-

Hmatch

Number of unsatisfiable classes / all classes

	CRS	PCS	CMT	CONFTOOL	SIGKDD	EKAW
CRS	-	9 / 38	11 / 44	0 / 53	19 / 64	42 / 87
PCS	20 / 38	-	0 / 53	21 / 62	0 / 73	10 / 97
CMT	13 / 44	0 / 53	-	35 / 68	0 / 79	n.a.
CONFTOOL	10 / 53	n.a.	26 / 68	-	4 / 88	21 / 112
SIGKDD	0 / 64	0 / 73	0 / 79	0 / 88	-	7 / 123
EKAW	47 / 87	33 / 97	18 / 103	1 / 112	42 / 123	-

Owl-CtxMatch

Number of unsatisfiable classes / all classes

	CRS	PCS	CMT	CONFTOOL	SIGKDD	EKAW
CRS	-	2 / 38	1 / 44	0 / 53	0 / 64	0 / 87
PCS		-	0 / 53	n.a.	0 / 73	0 / 97
CMT			-	18 / 68	n.a	0 / 103
CONFTOOL				-	0 / 88	39 / 112
SIGKDD					-	4 / 123
EKAW						-

In opposite to the other matching systems, OWL-CtxMatch generates not just equivalence correspondences, but also subsumption correspondences.

All used original ontologies, the mappings and the resulting ontologies are zipped in the file ontologies.zip from the following website:

`http://webrum.uni-mannheim.de/math/lski/ontdebug/ontologies.zip`

If you extract the contents of this file, you will get the following directory structur:

```
CMT.owl
CRS.owl
```

```
... (all other original ontologies)
coma-mappings/   (the coma mappings in txt-format)
coma-joined/     (contains the union-ontologies based on
          the coma mappings)
coma-evaluation/ (the evaluated coma mappings in txt-format)
... (the same directories for the other matching system)
```

3.4.2 Inconsistency created by Ontology Learning

3.4.2.1 PROTON

As described in Chapter Introduction, Völker and her colleagues made an experiment of human annotation of disjointness[Völker *et al.*, 2007b]. In this experiment, a large number of manually created disjointness are manually created. As a basis for the creation of the datasets and as background knowledge for the ontology learning algorithms they took a subset of the freely available PROTON ontology. The PROTON Ontology (PROTo ONtology) has been developed in the scope of the SEKT Project [8]. PROTON is a basic upper-level ontology to facilitate the use of background or preexisting knowledge for automatic metadata generation. PROTON covers the general concepts necessary for a wide range of tasks, including semantic annotation, indexing, and retrieval of documents. PROTON consists of 4 modules: System module, Top module, Upper module, and Knowledge Management module. In these experiments, the selected subset of PROTON contains 266 classes, 77 object properties, 34 datatype properties and 1388 siblings. Each concept pair was randomly assigned to 6 different people - 3 from each of two groups the first one consisting of PhD students from their institute (all of them professional ”ontologists”), the second is being composed of under-graduate students without profound knowledge in ontological engineering. Each of the annotators was given between 385 and 406 pairs along with natural language descriptions of the classes whenever those were available. Possible taggings for each pair were ’+’(disjoint), ’-’ (not disjoint) and ? (unknown). Furthermore, they computed the majority votes for all the above mentioned datasets by considering the individual taggings for each pair. If at least 50 percent (or 100 percent respectively) of the human annotators agreed upon ’+’ or ’-’ this decision was assumed to be the majority vote for that particular pair. Adding those created disjointness to Proton results in inconsistent PROTON ontologies. There are 24 unsatisfiable concepts in the PROTON ontology with the disjointness are created by 50 percent votes by Students. In this experiment, there are 6 data sets which contain inconsistencies. Contributor: Johanna Völker (University of Karlsruhe)

[8] http://proton.semanticweb.org/

3.4.2.2 KM1500

The 'km1500' ontology has been generated automatically from 1500 abstracts of the 'knowledge management' information space which is part of the BT Digital Library. We applied the ontology learning framework Text2Onto[Cimiano and Völker, 2005] in order to extract concepts, instances, taxonomic and non-taxonomic relationships, as well as disjointness axioms from these documents. However, unlike in our previous experiments on ontology learning and consistent ontology evolution[Haase and Völker, 2005], we omitted all confidence and relevance values generated by Text2Onto. Obviously, this gave us a bigger, but less reliable set of axioms resulting in a very high number of unsatisfiable concepts. Contributor: Johanna Vlker (University of Karlsruhe).

The 'km1500' ontology is converted into one in the DIG format with only disjoint axioms and subclass (impliesc) axioms. It leads to three sub-ontologies which are inconsistent. Contributor: Zhisheng Huang (Vrije University of Amsterdam)

Numbers of unsatisfiable classes

Ontology	Concepts	UC	Axioms	Size(KB)	DA	SA
km1500a	2116	879	3091	315	1915	1176
km1500b	2137	941	2937	302	1707	1230
km1500c	2724	1270	4265	424	2501	1764

UC= Unsatisfiable Concepts, DA = Disjoint Axioms, SA= Subclass Axioms

From the original km1500 ontology, we created N (N = 3, 5, 10) subsets of increasing size. Starting with the most reliable ontology learning results we subsequently added more elements (i.e. concepts, instances, relations and axioms) to the ontology by lowering the threshold. That lead to the following variants of the km1500 ontology. Contributor: Johanna Völker (University of Karlsruhe).

km1500_3.zip (3 data sets)
km1500_5.zip (5 data sets)
km1500_10.zip (10 data sets)

Those variants of the 'km1500' ontology are converted into ones in the DIG format with only disjoint axioms and subclass (impliesc) axioms. It leads to 10 sub-ontologies which are inconsistent. Contributor: Zhisheng Huang (Vrije University of Amsterdam)

km1500_3.dig.zip (2 inconsistent data sets)
km1500_5.dig.zip (3 inconsistent data sets)
km1500_10.dig.zip (5 inconsistent data sets)

Ontology	Concepts	UC	Axioms	Size(KB)	DA	SA
km1500_3ontology1	6383	0	5383	556	1	5382
km1500_3ontology2	9659	3657	10106	1035	1447	8659
km1500_3ontology3	9725	3991	10944	1116	2091	8853
km1500_5ontology1	4026	0	3291	339	0	3291
km1500_5ontology2	7692	0	6583	681	0	6583
km1500_5ontology3	9574	2999	9221	947	688	8533
km1500_5ontology4	9725	3991	10930	1115	2091	8839
km1500_5ontology5	9725	3991	10944	1116	2091	8853
km1500_10ontology1	1787	0	1331	137	0	1331
km1500_10ontology2	3665	0	2975	307	0	2975
km1500_10ontology3	5527	0	4621	477	0	4621
km1500_10ontology4	7351	0	6266	647	0	6266
km1500_10ontology5	9127	0	7910	817	0	7910
km1500_10ontology6	9573	2999	9218	947	685	8533
km1500_10ontology7	9714	3660	10650	1088	1991	8659
km1500_10ontology8	9725	3991	10930	1115	2091	8839
km1500_10ontology9	9725	3991	10937	1115	2091	8846
km1500_10ontology10	9725	3991	10944	1116	2091	8853

UC= Unsatisfiable Concepts, DA = Disjoint Axioms, SA= Subclass Axioms

Figure 3.8: km1500 ontologies

The km1500 ontologies and their scale data are shown in Figure 3.8. See the table, we can see that the series of the ontology 'km1500_10ontology' provides a rich variant of ontologies in this data set: the ontology 'km1500_10ontology10' is the largest scale ontology with 9725 concepts, 10944 axioms, and 3991 unsatisfiable concepts, whereas the ontology 'km1500_10ontology1' is a medium-scale ontology with 1787 concepts and 1331 axioms.

Chapter 4

Benchmarking Experiments

In this chapter, we report the experiments and the evaluation on various methods of processing inconsistent ontologies with respect to the following perspectives:

- Syntactic Approaches versus Semantic Approaches.
- Linear Extension versus Multi-Step Extension (k-extension).
- Reasoning with Inconsistent Ontologies versus Debugging of Inconsistent Ontologies.

Moreover, we will evaluate various description logic reasoners with inconsistent ontologies. We evaluate various approaches of processing inconsistent ontologies with respect to the three factors:

- **Quality of Query Answers**: We compare the system's query answers with the expected answers in a gold standard.
- **Performance**: We want to see how much the average time cost per query is for each method.
- **Scalability**: We want to know whether or not the system can handle large scale of ontologies.

4.1 Syntactic Approaches versus Semantic Approaches

Given that the mixed approach (using syntactic relevance for growing the relevant set, and using semantic relevance for backtracking, possibly using α-cuts) seems to be the best alternative to the purely syntactic approach of our earlier work, our experiment is

Ontology	Approach	α	IA	CA	RA	CIA	IA Rate(%)	IC Rate(%)
100_rudis	FMC	n/a	266	219	32	12	50.28	91.68
100_rudis	SD	0.75	225	260	32	12	42.53	91.68
100_rudis	SD	0.80	239	246	32	12	45.18	91.68
100_rudis	SD	0.85	246	238	32	13	46.50	91.49
50_studis	FMC	n/a	292	179	44	14	55.20	89.04
50_studis	SD	0.75	234	248	38	9	44.23	91.12
50_studis	SD	0.80	246	230	39	14	46.503	89.98
50_studis	SD	0.85	254	219	41	15	4802	89.41

IA = Intended Answers, CA = Cautious Answers, RA = Reckless Answers, CIA = Counter-Intuitive Answers, IA Rate = Intended Answers(%), IC Rate = IA+CA(%), FMC = First Maximal Consistent subset, SD = Semantic Distance, α=Threshold, 50_studis = the ontology 'proton_50_studis.xml', 100_rudis = the ontology 'proton_100_rudis.xml',

Figure 4.1: Syntactic approach and Semantic approach by α cutting-off.

Ontology	Approach	α	TimeCost	TimeRatio	IA Change Rate
100_rudis	FMC	n/a	114.63	n/a	n/a
100_rudis	SD	0.75	22.37	5.12	-0.15
100_rudis	SD	0.80	39.96	2.87	-0.10
100_rudis	SD	0.85	54.28	2.11	-0.08
50_studis	FMC	n/a	175.27	n/a	n/a
50_studis	SD	0.75	23.16	7.57	-0.20
50_studis	SD	0.80	39.39	4.45	-0.16
50_studis	SD	0.85	60.43	2.90	-0.13

TimeCost = Time Cost per Query (second), TimeRatio = TimeCost(FMC)/TimeCost(SD), IR Rate Change = (IA(SD)-IA(FMC))/IA(FMC)

Figure 4.2: Syntactic approach and Semantic approach by α cutting-off: Time Performance

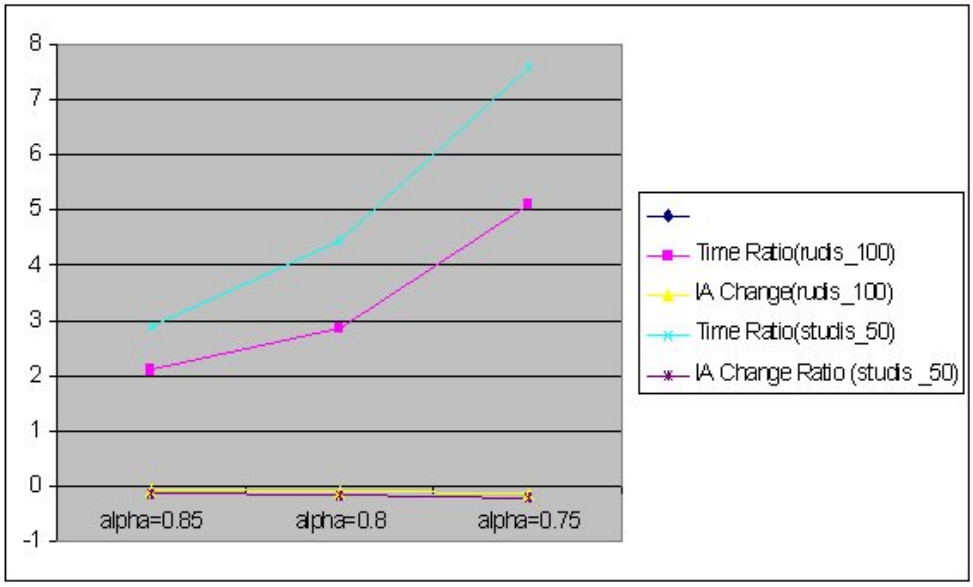

Figure 4.3: α Cutting-off and Time Performance

aimed at (1) finding out the quality of the answers generated by the mixed approach, and (2) finding out the quality/cost trade-offs that can be obtained by varying the α-levels.

We created 529 subsumption queries randomly, and obtained PION's answers of these queries with backtracking done either blindly (First Maximal Consistent Subset, FMC), or via the semantic distance (SD). We compared these answers against a hand-crafted Gold Standard that contained the humanly-judged correct answer for all of these 529 queries. For each query, the answer given by PION can be classified in one of the following categories, based on the difference with the intuitive answer in the Gold Standard: Intended Answer (IA), Cautious Answer (CA), Reckless Answer (RA), and Counter-Intuitive Answer (CIA), as defined in the previous chapter. Obviously, one would like to maximize the Intended Answers, and minimize the Reckless and Counter-intuitive Answers.

Furthermore, We introduced different α-thresholds in the Over-determined processing to see how the tradeoff between the quality of query-answers and the time performance is effected by different cutting levels.

Our results obtained by running PION with the data and the tests described above are shown in Figure 4.1 and Figure 4.2. The first 4 rows show experiments on the proton_rudis_100 ontology, the final 4 rows on the proton_studis_50 ontology. In all cases, we use use syntactic relevance for growing the relevance set until an answer can be found, but they differ on what happens when the relevance set becomes inconsistent, and backtracking is required. On the first line (labeled FMC, for First Maximal Consistent subset), the backtracking is done blindly, on the other lines, backtracking is is guided by the semantic distance function, at different α-levels (i.e. with different sizes of the backtracking steps; smaller values for α, i.e. lower thresholds, means that more formulas are removed during

backtracking).

Not listed in the table is the fact that among the 529 queries, 414 (i.e. 78%) resulted in relevance sets that became inconsistent before the query could be answered meaningfully, hence they needed a backtracking phase.

The tables shows that when switching from syntactic backtracking (labeled FMC) to semantic backtracking (labeled SD) the intended answer (IA) rate does indeed drop, as predicted in section 2.7.2. Furthermore, the IA-rate declines slowly with decreasing α-levels. Similarly, the cautious answer rate increases slowly with decreasing α-levels. This is again as expected: larger backtracking steps are more likely to remove too many formulas from the relevance set, hence potentially making the relevance set too small. Or put another way: the hill-climbing search performed in the ODP phase is aiming to get close to a maximal consistent subset, but larger hill-climbing steps make it harder to end up close to such a set, because of possible over-pruning.

The combined IC-rate (combining intended and cautious answers, i.e. those answers that are not incorrect, but possibly incomplete), stays constant across between and FMC and SD, and across all α-levels.

It is important to note that the numbers of reckless and counter-intuitive answers remains constant[1]. This means that although the semantically guided large-step reductions (at low α-levels) do of course remove formulas, they do not remove the wrong formulas, which could have potentially lead to reckless or counter-intuitive answers.

Summarizing, *when switching from FMC to SD, and with decreasing α-levels, the completeness of the algorithm (IA Rate) gradually declines, while the soundness of the algorithm (IC rate) stays constant.*

Although these findings on the answer quality are reassuring (the semantic backtracking doesn't damage the quality), they are not by themselves a reason to prefer semantic backtracking over syntactic backtracking. The strong point of the semantic backtracking becomes clear when we look at the computational costs of syntactic and semantic backtracking, particularly in the light of the answer quality.

Above, we have seen that the answer quality only degrades very gradually with decreasing α-levels. The final two columns of table 4.2 however show that the answer *costs* reduce dramatically when switching from syntactic to semantic backtracking, and that they drop further with decreasing α-levels. The absolute computation time is more than halved when switching from FMC to SD ($\alpha = 0.85$), and is again more than halved when dropping α from 0.85 to 0.75, leading to an overall efficiency gain of a factor of 5. Of course, this efficiency is gained at the cost of some loss of quality, but this loss of quality (the drop in completeness, the IA rate) is very modest: the twofold efficiency gain at $\alpha = 0.85$ is gained at a price of a drop of only 3 percentage points in completeness, and the fivefold efficiency gain at $\alpha = 0.75$ is gained at a price of a drop of only 7 percentage points in completeness. (Remember that the soundness of the results, the IC rate, is not at

[1] and is even going significantly down in the proton_studis_50 case.

Ontology	SelectionFunction	ODP	α	Queries	IA	CA	RA	CIA
100_studis	Syn	FMC	n/a	529	234	249	33	13
100_studis	SynCon	SD	1.00	529	189	309	22	9

Syn = The syntactic relevance based selection function, SynCon = The direct concept relevance based selection function.

Figure 4.4: Selection Functions and Over-determined Processing: Quality of Query Answers

Ontology	SelectionFunction	ODP	α	TimeCost	TimeRatio	IA Change Rate
100_studis	Syn	FMC	n/a	91.30	45.05	n/a
100_studis	SynCon	SD	1.00	94.14	28.11	1.62

Syn = The syntactic relevance based selection function, SynCon = The directly concept relevance based selection function.

Figure 4.5: Selection Functions and Over-determined Processing: Time Performance

all affected). The chart of α Cutting-off and its time performance is shown in Figure 4.3.

An additional experiment on the comparison between the blind backtracking and the informed backtracking is that we use the direct concept relevance based selection function which is described in Chapter 2 to enhance the informed backtracking without the α cutting off (i.e., $\alpha = 1.00$). The test results are shown in Figure 4.4 and Figure 4.5. From the tables we can see that the semantic approach for informed backtracking with the enhanced direct concept relevance based selection function can reduce the amount of counter intuitive answers (CIA) significantly with a stable IC rate (94.14%) and a decreased amount of intuitive answers (33.75% from 44.23%). Moreover, its time performance is improved by 162%.

Summarizing, *semantic backtracking with cut-off levels yields a very attractive cost/quality trade-off between costs in terms of run-time, and the quality in terms of soundness and completeness of the answers.*

4.2 Linear extension versus Multi-step extension

In this experiment we want to see how k-extension with different k value may change the quality of query answer and the time performance for reasoning with inconsistent ontologies.

First we conduct three experiments on the ontology proton_50_rudis with different k values in the syntactic relevance based selection function with the blind backtracking approach (i.e., FMC). Note that the linear extension strategy is a k extension one with

Ontology	Approach	α	k	IA	CA	RA	CIA	IA Rate(%)	IC Rate(%)	TC
50_rudis	FMC	n/a	1	236	245	36	12	44.61	90.93	47.64
50_rudis	FMC	n/a	2	243	232	37	17	45.94	89.79	49.22
50_rudis	FMC	n/a	3	338	141	37	13	63.89	90.55	59.21
50_rudis	SD	0.75	2	198	288	35	8	37.43	91.87	20.53
50_rudis	SD	0.80	2	212	268	36	13	40.08	90.74	32.55
50_rudis	SD	0.85	2	226	253	36	14	42.72	90.55	43.44
50_rudis	SD	0.75	3	201	291	31	6	38.00	93.01	156.29

IA = Intended Answers, CA = Cautious Answers, RA = Reckless Answers, CIA = Counter-Intuitive Answers, IA Rate = Intended Answers(%), IC Rate = IA+CA(%), FMC = First Maximal Consistent subset, SD = Semantic Distance, α=Threshold, TC=Time cost per query (second)

Figure 4.6: K extension and α cutting-off.

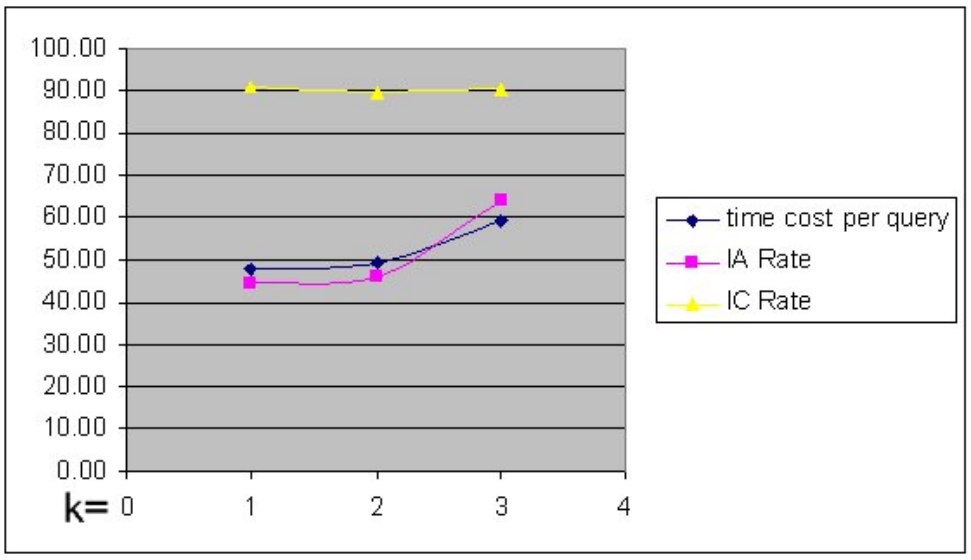

Figure 4.7: K extension and Performance

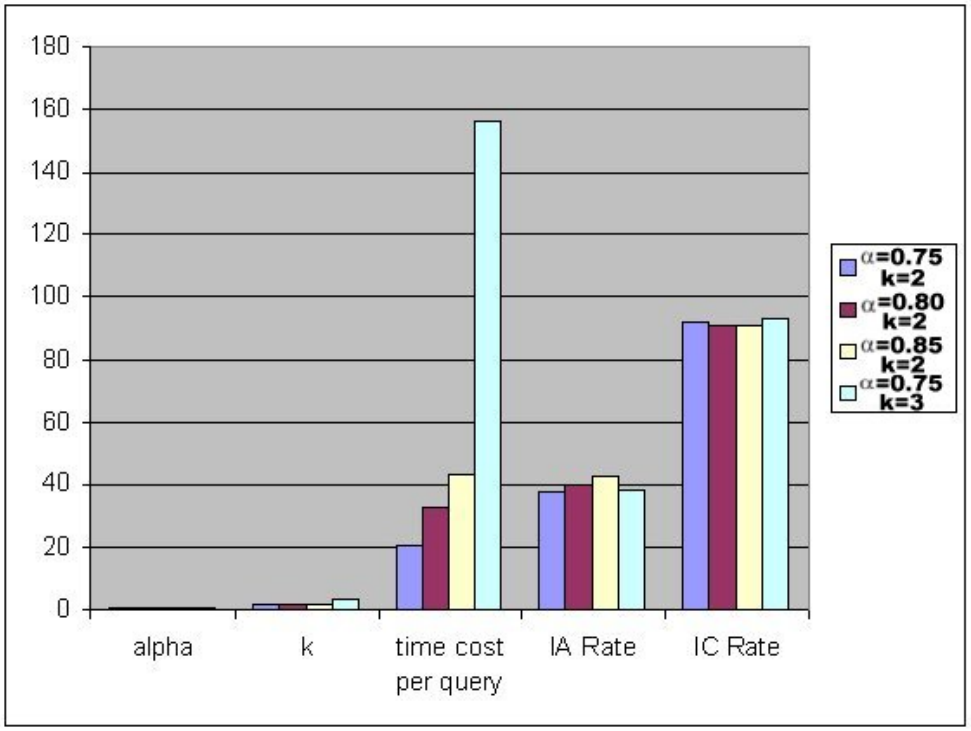

Figure 4.8: K extension and Performance

k=1. The results are shown in the first three rows of Figure 4.6. Frow the data in the table we know that the amount of intended answers (IA) increases (from 44.61% into 63.89%) when the k-value grows up (from 1 into 3). However, its time performance increases(from 47.61 second per query into 59.21 second per query) when the k value is changed from 1 into 3.

The second group of the experiments on the multi-step extension is that we test the same ontology with different k values in the informed backtracking approach (i.e., SD) under different α cutting-off. The results are shown in the four rows in Figure 4.6. In the experiments we observe the following facts:

- When the k value is fixed (i.e., $k = 2$), the amount of intended answers increases (from 37.43% into 42.72%) when the α cutting off increases (from 0.75 into 0.85), however, with an increased time cost (from 20.53 second per query into 43.44 second per query).
- When the k value is changed (from k=2 into k=3), the amount of intended answers decreases slightly (from 37.43% into 38%), however, the amount of counter intuitive answers also decreases (from 8 into 6) and the time cost increases significantly (from 22.53 second per query into 156.29 second per second).

The first fact tells us that the amount of intended answers will be increased by decreasing

an α value. We observe that the same effect occurs in the experiment of the informed backtracking in the linear extension. The second fact shows that a bigger k value would be more expensive with respect to its time performance, although it may improve the quality of query answers. The relationship among k values, α cutting off, the quality of query answers, and the time performance is also shown as a chart in Figure 4.8.

Summarizing, *when increases the k-value under FMC, the intended answers increases with the increasing time cost slightly. When switching from FMC to SD with a non-trivial k value (i.e., $k > 1$) under an α-cutting-off, the completeness of the algorithm (IA Rate) gradually declines, while the soundness of the algorithm (IC rate) increases or stays stable. That means that the multi-step extension does not help too much with the informed backtracking.*

4.3 Reasoning with Inconsistent Ontologies versus Debugging of Inconsistent Ontologies

In this section we want to compare the results obtained by using the approaches of reasoning with inconsistent ontologies with the results obtained by using debugging approaches, although those two kinds of approaches are motivated by different application scenarios. The former approach is designed as ones for run time, whereas the latter approach is designed as ones for design time. Moreover, debugging approaches usually require some intervention by human knowledge or a ranking on the results by manual. Therefore, it does not make sense if we compare the results by those approaches with respect to time performance. However, we still want to know what the differences between the reasoning approach and the debugging approach are with respect to the quality of query answers.

The first difficulty we encounter for this comparison is that those two kinds of approaches use different answer values. The answer of the systems of reasoning with inconsistent ontologies is a multi-value set: $\{accepted, rejected, undetermined, over-determined\}$, whereas debugging inconsistent ontologies usually result in a set of consistent ontologies. We use the standard reasoners on those repaired ontologies. Thus, the answer value set of queries on ontologies which result from the debugging approach is the standard boolean answers by most DL reasoners, i.e, $\{true, false\}$.

In order to compare the approach of reasoning with inconsistent ontologies with the approach of debugging, we should build a mapping between those two different answer value sets. It is clear that the answer 'true' is interpreted as 'accepted'. However, the answer 'false' can be interpreted as 'rejected', or as 'rejected or undetermined'. In this document we consider both the interpretations. The interpretation I is defined as one in which 'false' is interpreted as 'rejected', whereas The interpretation II is defined as one in which 'false' is interpreted as 'rejected or undetermined'.

We use the DION system to obtain the repair information (i.e., MUPS, MIPS, and

pinpoints, etc.) for the ontology 'proton_100_rudis.dig.xml' and the ontology 'proton_50_studis.dig.xml'. That results in too many possibilities to repair the ontologies. We remove some axioms from the ontologies manually by the intervene of human knowledge. Finally, we pick up two consistent sub-ontologies for each repaired ontology: {100_rudis_1.dig.xml, 100_rudis_2.dig.xml, 50_studis_1.dig.xml, 50_studis_2.dig.xml}, which are considered as the most intuitive results of the repair.

Meanwhile we also use the RepairTab and the RaDON to repair the ontology 'proton_100_rudis.dig.xml' and the ontology 'proton_50_studis.dig.xml'[2]. We obtain a consistent sub-ontology '100_rudis_abdn.dig.xml' by using the RepairTab for the ontology 'proton_100_rudis.dig.xml' However, the RepairTab fails to resolve the unsatisfiable concept 'Lock' in the ontology 'proton_50_studis.dig.xml'. The RepairTab uses a set of heuristics to rank the problematic axioms with respect to an unsatisfiable concept. The concept "Lock" is unsatisfiable due to three axioms, which two of the axioms were ranked equally by the system. It is because the set of heuristics failed to access the ranking of the two axioms. For the rest of the concepts, the heuristics were applied to rank the axioms, whose value is ranged [-1, 1]. The axiom with the lowest ranking will be removed.

Using the RaDON system, we obtain a consistent sub-ontology '100_rudis_radon.dig.xml' for the ontology 'proton_100_rudis.dig.xml' and a consistent sub-ontology '50_studis_radon.dig.xml' for the ontology 'proton_50_studis.dig.xml'.

The comparison between the approach of reasoning with inconsistent ontologies and various systems of debugging inconsistent ontologies is show in Figure 4.9. The tests show that the debugging approaches, the DION system, the RepairTab system, and the RaDON system, can gain higher rates of intended answers (IA) for both the interpretations I and II (81.85% to 50.28% at least). However, if we consider the IC Rate, there are no big differences between the debugging approaches and the reasoning approaches, which depend on what kind of the interpretation we take. If we take the interpretation I, the reasoning approaches gain higher IC rates (95.65% to 81.85% at least). If we take the interpretation II, the debugging approaches may gain higher IC rates (96.41% to 89.044%).

From the table we see that various debugging approaches, the DION system, the RepairTab system, and the RaDON system, can gain almost the same quality of query answers, although the RepairTab system fails to resolve the inconsistency of the ontology 'proton_50_studis'.

Summarizing, *Debugging approach gains higher quality of answers than the approach of reasoning with inconsistent ontologies; however, it needs some intervention by human knowledge*.

[2]We will report this benchmarking experiment on the debugging approaches only with the systems DION, RepairTab, and RaDON, because the other debugging systems are not available for public

Ontology	System	IA	CA	RA	CIA	IA Rate(%)	IC Rate(%)
100_rudis_1	DION(I)	433	0	76	20	81.85	81.85
100_rudis_1	DION(II)	498	8	11	12	94.14	95.65
100_rudis_2	DION(I)	432	0	76	21	81.66	81.66
100_rudis_2	DION(II)	497	9	11	12	93.95	95.65
100_rudis	RepairTab(I)	433	0	76	20	81.85	81.85
100_rudis	RepairTab(II)	498	8	11	12	94.14	95.65
100_rudis	RaDON(I)	433	0	76	20	81.85	81.85
100_rudis	RaDON(II)	498	9	11	11	94.14	95.67
100_rudis	PION(Syn)	266	219	32	12	50.28	91.68
50_studis_1	DION(I)	427	0	76	26	80.72	80.72
50_studis_1	DION(II)	495	15	8	11	93.57	96.41
50_studis_2	DION(I)	431	0	76	22	81.47	81.47
50_studis_2	DION(II)	499	11	8	11	94.33	96.41
50_studis	RaDON(I)	431	0	76	22	81.47	81.47
50_studis	RaDON(II)	499	11	8	11	94.33	96.41
50_studis	PION(Syn)	292	179	44	14	55.20	89.04

DION(I)= Using DION as a system of debugging inconsistent ontologies with the interpretation I(i.e., false =rejected). DION(II)= Using DION as a system of debugging inconsistent ontologies with the interpretation II (i.e., false = rejected or undetermined). RepairTab(I)=Using the RepairTab system with the interpretation I, RepairTab(II)=Using the RepairTab system with the interpretation II, RaDON(I)=Using the RaDON system with the interpretation I, RaDON(II)=Using the RaDON system with the interpretation II, PION(Syn) = Using PION as a system of reasoning with inconsistent ontologies with the syntactic approach. IA = Intended Answers, CA = Cautious Answers, RA = Reckless Answers, CIA = Counter-Intuitive Answers, IA Rate = Intended Answers(%), IC Rate = IA+CA(%)

Figure 4.9: Reasoning vs. Debugging

Ontology	Concepts	UC	Axioms	Racer	Pellet	FACT++
km1500_10_1	1787	0	1331	2.719	2.047	0.703
km1500_10_2	3665	0	2975	6.282	4.188	1.469
km1500_10_3	5527	0	4621	10.125	7.485	2.313
km1500_10_4	7351	0	6266	14.047	11.422	3.125
km1500_10_5	9127	0	7910	17.218	15.469	3.765
km1500_10_6	9573	2999	9218	296.391	17.813	4.188
km1500_10_7	9714	3660	10650	697.375	79.312	5.125
km1500_10_8	9725	3991	10930	368.781	132.344	4.875
km1500_10_9	9725	3991	10937	1019	135.797	4.813
km1500_10_10	9725	3991	10944	1843.77	135.515	4.968

UC=Unsatisfibale Concepts, km1500_10_n =km1500_10/ontology_n

Figure 4.10: Reasoners and Performance

4.4 Inconsistency Processing and Reasoners

In this section we will report an experiment on various DL reasoners to see their performance and scalability with respect to the detection of all unsatisfiable concepts. The ontologies used for this experiment are obtained from the km1500 ontology, which was divided into 10 different levels of subontologies, as described in the previous chapter. The tested ontologies range from medium sized consistent ontology (with 1787 concepts and 1331 axioms) to large scale inconsistent ontologies (with 9726 concepts, 10944 axioms, and 3991 unsatisifiable concepts).

The performance and the scalability of various DL reasoners (Racer, Pellet, and FACT++[3]) is shown in Figure 4.10. The tests show that Racer behaves very well for consistent ontologies, however, it goes worst when it encounters large scale inconsistent ontologies. It takes only 2.719 second to detect a medium sized ontology (with 1787 concepts and 1331 axioms) to conclude that it is consistent. However, it takes 296.391 seconds (almost 100 times on the time cost) to handle a large scale inconsistent ontology (with 9573 concepts, 9218 axioms, and 2999 unsatisfiable concepts). It behaves much worse (it takes 1843.77 seconds) when it handles a larger scale inconsistent ontology (with 3991 unsatisfiable concept). The FACT++ is the winner of the performance and the scalability. It takes only 4.968 seconds to detect all of unsatisfiable concepts (3991 unsatisfiable concepts) in a large scale ontology (with 9725 concepts and 10944 axioms). The Pellet behaves not too bad with respect to the performance and the scalability. It takes 135.515 seconds to handle a large scale inconsistent ontology. The difference of the performance and the scalability of those Dl reasoner is also shown as a chart in Figure 4.11.

[3]KAON2 fail to pass those tests. Therefore, it is absent from the table.

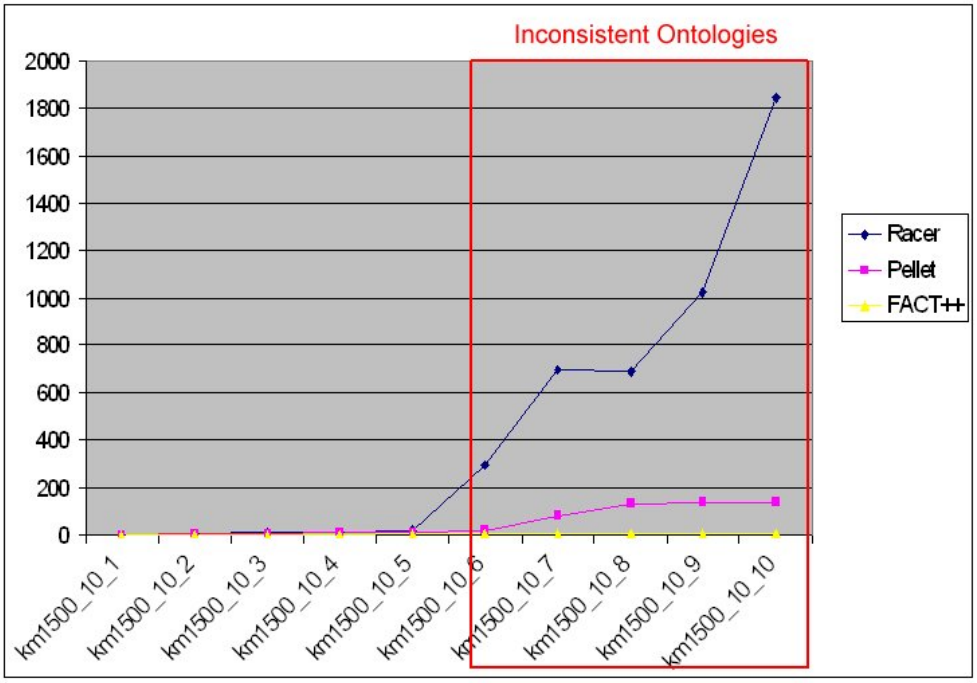

Figure 4.11: Reasoners, Inconsistent Ontologies, and Time Cost per Query(second)

We considered the experiment described in this section as a black-box benchmarking one, because we would not detect the internal structure and analyze the implementation methods used in those DL reasoners to see the reasons why they behave so differently.

Summarizing, *FACT++ is the best DL reasoner with inconsistent ontologies with respect to its time performance and the scalability.*.

Chapter 5

Discussions and Conclusions

5.1 Discussions

We have performed a series of benchmarking experiments of processing inconsistent ontologies with realistic ontologies. In this chapter we discuss the implications of those benchmarking experiments and its relation with the approaches used in other relevant areas, such as the similarity measure in linguistics and processing inconsistency in database systems. Based on the analysis and the discussion, we suggest the future work of processing inconsistent ontologies.

5.1.1 Semantic Approach for Reasoning with Inconsistent Ontologies

Research from a number of different areas is relevant to the current work of the semantic approach proposed in this document. Semantic distances and similarity measures have been widely used in computational linguistics [Budanitsky and Hirst, 2001, Lin, 1998] and ontology engineering [Haase, 2006, Maedche and Staab, 2002]. [Gligorov *et al.*, 2007] proposes the use of a Google-based similarity measure to weigh approximate ontology matches. Our research is the first attempt to introduce the Google Distance for reasoning with inconsistent ontologies. In essence we are using the implicit knowledge hidden in the Web for explicit reasoning purposes.

The main contributions of our semantic approach are: a) we investigated how a semantic relevance-based selection function can be developed by using information provided by a search engine, in particular, by using the Normalized Google Distance; b) we provided variants of backtracking strategies for reasoning with inconsistent ontologies, and c) we showed that semantic distances can be used for handling large scale ontologies through a tradeoff between run-time and the degree of incompleteness of the algorithm.

In our experiment we applied our PION implementation to realistic test data. The

experiment used a high-quality ontology that became inconsistent after adding disjointness statements that had the full support of a group of experts. The test showed that the run-time of informed semantic backtracking is much better than that of blind syntactic backtracking, while the quality remains comparable. Furthermore the semantic approach can be parametrised so as to stepwise further improve the run-time with only a very small drop in quality.

5.1.2 Integrating Reasoning with Inconsistent Ontologies with Debugging of Inconsistent Ontologies

In this document we have performed several experiments for the comparison between the approaches of reasoning with inconsistent ontologies and the approaches of debugging inconsistent ontologies. Various debugging systems, including DION (Amsterdam), RepairTab (Aberdeen), and RaDON (Karlsruhe), have been examined and tested. The tests show that debugging approaches can gain higher amount of intended answers with some kinds of intervene by human knowledge, like picking up repaired axioms manually or ranking on axioms.

From the experiments we also observe the fact that there are no big differences of the quality of query answers by the three tested debugging systems (DION, Repair, RaDON). One of the explanations for it is that they use the similar ideas, like MUPS and MIPS, for the debugging.

One of the future work is to integrate a system of reasoning with inconsistent ontologies with a debugging system. Namely, we use a debugging system to attempt some repair of inconsistent ontologies automatically or semi-automatically first. That may lead to some part of inconsistent ontologies which are difficult to be repaired. Then we apply a system of reasoning with inconsistent ontologies to get meaningful answers from this partially repaired inconsistent ontologies.

5.1.3 Processing Inconsistencies in Database Systems

Processing Inconsistencies have been well studied in Database systems[Arenas *et al.*, 1999, Fuxman *et al.*, 2005, Staworko *et al.*, 2006, Fuxman and Miller, 2007]. Integrity constraints have long been used to maintain data consistency. Therefore, one of the main issues of resolving inconsistencies in database systems is to maintain the integrity constraints. Namely, a data base system is considered to be consistent one if it satisfies given integrity constraints.

In [Arenas *et al.*, 1999], Arenas et al. consider the problem of the logical characterization of the notion of consistent answer in a relation data that may violate given integrity constraints. A method for computing consistent answers is proposed with the minimal repair on the inconsistent databases. The method is based on an iterative procedure whose

termination for several classes of constraints is proved. The idea of the minimal repair in this approach is similar with that used in the approaches of debugging inconsistent ontologies.

In [Fuxman *et al.*, 2005], Fuxman et al. present ConQuer, a system for efficient and scalable answering of SQL queries on databases that may violate a set of constraints. ConQuer permits users to postulate a set of key constraints together with their queries. The ConQuer system rewrites the queries to retrieve all data that is consistent with respect to the constraints. In [Fuxman and Miller, 2007], Fuxman et al. concentrate on the first-order query rewriting for inconsistent databases. They present an algorithm that computes the consistent answers for a large and practical class of conjunctive queries by returning a first-order query such that for every database, the consistent answers for a query can be obtained by evaluating the first-order query directly on the database. This approach suggests an interesting approach for reasoning with inconsistent ontologies, in which a consistent sub-ontology for a given query is selected to obtain meaningful answers. The similar idea can be used for reasoning with inconsistent ontologies is to allow users to postulate a set of axioms together with their queries. Namely, users can claim a set of axioms which are considered as a kernel of the ontology with respect to given queries, so that meaningful answers can be obtained from this pre-selected set.

We should notice that application scenarios and environments between database systems and ontology engineering are quite different. The Closed World Assumption (CWA) is usually accepted in database systems, whereas the Open World Assumption (OWA) is usually assumed in ontology engineering. Furthermore, in database systems, given integrity constraints are supposed to be satisfiable. Namely, they are already known to be consistent at design time. However, for inconsistent ontologies, we would not assume that some consistent sub-ontologies have been known by users or systems without any reasoning.

5.2 Future Work

As we discuss in Chapter 3, benchmarking is a continuous process for improving by systematically evaluating tested systems, and comparing them to those considered to be the best. Namely, benchmarking is a continuous processing of evaluation. What we report in this document is just the first step of continuous evaluation of various methods for processing inconsistent ontologies. That is the reason why develop the benchmarking suite for its sustainability. We expect to use this benchmarking suite for further evaluation of various methods/systems of processing inconsistent ontologies in other projects in the future.

Moreover, we will continue to improve the systems we have developed, including the system PION and DION after this comprehensive evaluation with respect to various benchmarking metrics (i.e., quality of query answers, performance, and scalability).

There have been several main future work for the system PION and DION. One of the future tasks is to make the NGD (Normalized Google Distance) component well integrated with the architecture of PION, so that the NGD values can be dynamically obtained at run time, rather than as the pre-loaded libraries, as it is done in the present implementation.

One of the future work for reasoning with inconsistent ontologies is to use other semantic distance measure rather Google distance, like using WordNet or information provided by other search engines such as Yahoo. It is also interesting to see how the same approach can be extended into the applications in which linguistic information or legacy information are not available for a similarity/distance measure. As discussed previously, the future work includes that the support for users to postulate a (consistent) set of axioms together with their queries and integrating a system of reasoning with inconsistent ontologies with a debugging system.

5.3 Concluding Remarks

The work reported in this document is the first comprehensive evaluation of various methods for processing inconsistent ontologies. More work of benchmarking for the improvement of various methods for processing inconsistent ontologies should be continued. The benchmarking suite which is implemented in this task provides a nice infrastructure for systematic evaluation of processing inconsistent ontologies. It is also very significant for the sustainability. The semantic approach for reasoning with inconsistent ontologies by using Google distances deserves to be investigated deeply further, because the benchmarking experiments reported in this document show that the semantic approach can significantly improve reasoning performance over the syntactic approach. This methods allows to trade-off computational cost for inferential completeness, hence providing attractive scalability.

Bibliography

[Arenas *et al.*, 1999] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *ACM Symposium on Princiles of Database System (PODS)*, pages 68–79, 1999.

[Baader and Hollunder, 1995] F. Baader and B. Hollunder. Embedding defaults into terminological representation systems. *J. Automated Reasoning*, 14:149–180, 1995.

[Baader *et al.*, 2003] 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.

[Belnap, 1977] N. Belnap. A useful four-valued logic. In *Modern Uses of Multiple-Valued Logic*, pages 8–37, Dordrecht, 1977. Reidel.

[Budanitsky and Hirst, 2001] Alexander Budanitsky and Graeme Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources, 2nd meeting of the North American Chapter of the Association for Computational Linguistics*. Pittsburgh, PA., 2001.

[Chopra *et al.*, 2000] Samir Chopra, Rohit Parikh, and Renata Wassermann. Approximate belief revision- prelimininary report. *Journal of IGPL*, 2000.

[Cilibrasi and Vitany, 2007] R. Cilibrasi and P. Vitany. The google similarity distance. *IEEE ACM Transactions on Knowledge and Data Engineering*, 19:3:370–383, 2007.

[Cilibrasi and Vitanyi, 2004] Rudi Cilibrasi and Paul Vitanyi. Automatic meaning discovery using google, 2004.

[Cimiano and Völker, 2005] Philipp Cimiano and Johanna Völker. Text2onto - a framework for ontology learning and data-driven change discovery. In Andres Montoyo, Rafael Munoz, and Elisabeth Metais, editors, *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB),volume 3513 of Lecture Notes in Computer Science*, pages 227–238, 2005.

[Console and Dressler, 1999] Luca Console and Oskar Dressler. Model-based diagnosis in the real world: Lessons learned and challenges remaining. In *Proceedings of the*

sixteenth International Joint Conference on Artificial Intelligence, IJCAI'99, pages 1393–1400. 1999.

[d'Aquin *et al.*, 2007] M. d'Aquin, C. Baldassarre, L. Gridinoc, S. Angeletou, M. Sabou, and E. Motta. Characterizing knowledge on the semantic web with watson. In *5th International EON Workshop at ISWC/ASWC07*, 2007.

[de Kleer and Williams, 1987] J de Kleer and B C Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[de la Banda *et al.*, 2003] 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, ACM-SGPLAN'03*, pages 32–43. ACM, 2003.

[Flouris *et al.*, 2006] Giorgos Flouris, Zhisheng Huang, Jeff Z. Pan, Dimitris Plexousakis, and Holger Wache. Inconsistencies, negations and changes in ontologies. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[Friedrich and Shchekotykhin, 2005] Gerhard Friedrich and Kostyantyn M. Shchekotykhin. A general diagnosis method for ontologies. In *Proceedings of the 4th International Semantic Web Conference, ISWC'05*, volume 3729 of *LNCS*, pages 232–246, 2005.

[Fuxman and Miller, 2007] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *Journal of Comput. Syst. Sci*, 73(4):610–635, 2007.

[Fuxman *et al.*, 2005] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *ACM SIGMOD International Conference on Management of Data*, 2005.

[García-Castro *et al.*, 2005] Raúl García-Castro, Diana Maynard, Holger Wache, Doug Foxvog, and Rafael González Cabero. Specification of a methodology, general criteria, and benchmark suites for benchmarking ontology tools. Project Report D2.1.4, KnowledgeWeb, 2005.

[Gligorov *et al.*, 2007] Risto Gligorov, Zharko Aleksovski, Warner ten Kate, and Frank van Harmelen. Using google distance to weight approximate ontology matches. In *Proceedings of WWW 2007*, 2007.

[Greiner *et al.*, 1989] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in reiters theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

[Haase and Völker, 2005] Peter Haase and Johanna Völker. Ontology learning and reasoning - dealing with uncertainty and inconsistency. In Paulo C. G. da Costa, Kathryn B. Laskey, Kenneth J. Laskey, and Michael Pool, editors, *Proceedings of*

the Workshop on Uncertainty Reasoning for the Semantic Web (URSW), pages 45–55, 2005.

[Haase *et al.*, 2005] 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.

[Haase, 2006] Peter Haase. *Semantic Technologies for Distributed Information Systems*. PhD thesis at the Universitat Karlsruhe, 2006.

[Hameed *et al.*, 2003] A. Hameed, A. Preece, and D. Sleeman. Ontology reconciliation. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 231–250. Springer Verlag, 2003.

[Huang and van Harmelen, 2006] Z. Huang and F. van Harmelen. Reasoning with inconsistent ontologies: Evaluation. Project Report D3.4.2, SEKT, 2006.

[Huang and van Harmelen, 2007] Z. Huang and F. van Harmelen. Pion2: A system of reasoning with inconsistent ontologies. Project Report D3.4.3, SEKT, 2007.

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

[Huang *et al.*, 2004] 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.

[Huang *et al.*, 2005] 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.

[Huang *et al.*, 2006] Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with inconsistent ontologies: Framework, prototype, and experiment. In John Davies, Rudi Studer, and Paul Warren, editors, *Semantic Web Technologies: Trends and Research in Ontology-based Systems*, pages 71–93. John Wiley and Sons, Ltd., 2006.

[Kalyanpur *et al.*, 2005] A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler. Debugging unsatisfiable concepts in owl ontologies. *Journal of Web Semantics*, 3(4), 2005.

[Kalyanpur *et al.*, 2006] A. Kalyanpur, B. Parsia, B. Cuenca-Grau, and E. Sirin. Beyond axioms: Fine-grained justifications for arbitrary entailments in owl-dl. In *Description Logic workshop (DL'06)*, 2006.

[Kalyanpur, 2006] A. Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, Univ. of Maryland, June 2006.

[Lam *et al.*, 2006] Joey Sik Chun Lam, Jeff Z. Pan, Derek Sleeman, and Wamberto Vasconcelos. Ontology inconsistency handling: ranking and rewriting axioms. Technical report aucs/tr0603, University of Aberdeen, 2006.

[Lam, 2007] Joey Sik Chun Lam. *Methods for Resolving Inconsistencies in Ontologies*. PhD thesis, Dept. of Computing Science, University of Aberdeen, 2007.

[Lin, 1998] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of International Conference on Machine Learning, Madison, Wisconsin, July*, 1998.

[Maedche and Staab, 2002] Alexander Maedche and Steen Staab. Measuring similarity between ontologies. In *Proceedings of the European Conference on Knowledge Acquisition and Management(EKAW-2002)*, pages 251–263, 2002.

[Meyer *et al.*, 2006] Thomas Meyer, Kevin Lee, Richard Booth, and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic alc. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI'06*, 2006.

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

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

[Schaerf and Cadoli, 1995] Marco Schaerf and Marco Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:249–310, 1995.

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

[Schlobach and Cornet, 2003b] 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.

[Schlobach and Huang, 2005] Stefan Schlobach and Zhisheng Huang. Inconsistent ontology diagnosis: Framework and prototype. Project Report D3.6.1, SEKT, 2005.

[Schlobach and Huang, 2007] Stefan Schlobach and Zhisheng Huang. Inconsistent ontology diagnosis and repair. Project Report D3.6.3, SEKT, 2007.

[Schlobach *et al.*, 2006] Stefan Schlobach, Ronald Cornet, and Zhisheng Huang. Inconsistent ontology diagnosis: Evaluation. Project Report D3.6.2, SEKT, 2006.

[Schlobach *et al.*, 2007] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39:317–349, 2007.

[Schlobach, 2005a] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In *ESWC*, pages 226–240, 2005.

[Schlobach, 2005b] Stefan Schlobach. Diagnosing terminologies. In *Proceedings of the twentieth National Conference on Artificial Intelligence, AAAI'05*, pages 670–675, 2005.

[Staworko *et al.*, 2006] Slawomir Staworko, Jan Chomicki, and Jerzy Marcinkowski. Preference-driven querying of inconsistent relational databases. In *EDBT Workshops*, pages 318–335, 2006.

[Volker *et al.*, 2007a] Johanna Volker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning disjointness. In *Proceedings of ESWC2007*, 2007.

[Völker *et al.*, 2007b] Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning disjointness. In *Proceedings of the 4th European Semantic Web Conference (ESWC'07)*. Springer, 2007.

[Wielemaker *et al.*, to appear] Jan Wielemaker, Zhisheng Huang, and Lourens van der Meij. Swi-prolog and the web. *Journal of Theory and Practice of Logic Programming*, to appear.