



Reasoning with Inconsistent Ontologies: Framework and Prototype

**Zhisheng Huang, Frank van Harmelen, Annette ten Teije,
Perry Groot, and Cees Visser
(Vrije Universiteit Amsterdam)**

Abstract.

EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D3.4.1(WP3.4)

In this document we propose a general framework for reasoning with inconsistent ontologies. We present formal definitions of soundness, meaningfulness, local completeness, and maximal completeness of an inconsistency reasoner. We propose and investigate a pre-processing algorithm and discuss the strategies of inconsistency reasoning based on pre-defined selection functions dealing with concept relevance. In this document, we also present a prototype of a reasoner for Processing Inconsistent ONtologies (PION), which is implemented in XDIG, an extended DIG Description Logic Interface for Prolog. We also discuss how the syntactic relevance can be used in the implementation of the prototype.

Keyword list: ontology management, inconsistent ontologies, reasoning

Document Id. SEKT/2004/D3.4.1/v1.0
Project SEKT EU-IST-2003-506826
Date December 31, 2004
Distribution public

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.

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

Jozef Stefan Institute

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

University of Sheffield

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

Intelligent Software Components S.A.

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

Ontoprise GmbH

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

Vrije Universiteit Amsterdam (VUA)

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

Empolis GmbH

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

University of Karlsruhe, Institute AIFB

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

University of Innsbruck

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

Kea-pro GmbH

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

Sirma AI EAD, Ontotext Lab

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

Universitat Autònoma de Barcelona

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

Executive Summary

The classical entailment in logics is *explosive*: any formula is a logical consequence of a contradiction. Therefore, conclusions drawn from an inconsistent ontology by classical inference may be completely meaningless. An inconsistency reasoner is one which is able to return meaningful answers to queries, given an inconsistent ontology.

In this document, we overview reasoning with inconsistency in logics and AI, in particular, in paraconsistent logics and approximation reasoning. Furthermore, we examine several typical examples and scenarios of inconsistency in the context of the Semantic Web. We propose a general framework for reasoning with inconsistent ontologies. We present the formal definitions of soundness, meaningfulness, local completeness, and maximal completeness of an inconsistency reasoner. We propose and investigate a pre-processing algorithm, discuss the strategies of inconsistency reasoning based on pre-defined selection functions dealing with concept relevance.

In this document, we also present a prototype of a reasoner for Processing Inconsistent Ontologies (PION), which is implemented in XDIG, an extended DIG Description Logic Interface for Prolog. We also discuss the architecture of PION and show how the syntactic relevance can be used for the prototype.

Contents

1	Introduction	2
2	Inconsistency in the Semantic Web	4
2.1	Inconsistency by Mis-presentation of Default	4
2.2	Inconsistency Caused by Polysemy	5
2.3	Inconsistency through Migration from Another Formalism	6
2.4	Inconsistency Caused by Multiple Sources	6
3	Framework of Reasoning with Inconsistent Ontologies	8
3.1	Formal Definitions	8
3.2	Algorithms	10
3.2.1	Inconsistency Detection	10
3.2.2	Inconsistency Reasoning	11
3.3	Selection Function	14
4	Prototype of a Reasoner with Inconsistent Ontologies	17
4.1	Introduction	17
4.2	Architecture	17
4.3	Implementation	19
4.4	Functionalities	19
4.5	Installation	20
4.6	PION Testbed	22
5	Discussion and Conclusions	25

Chapter 1

Introduction

The Semantic Web is characterized by scalability, distribution, and multi-authorship. All these characteristics may introduce inconsistencies in the Semantic Web. Limiting language expressivity with respect to negation (like RDF and other languages that are based on negation as failure) can avoid inconsistencies to a certain extent. However, for specifications the expressivity of these languages is quite limited. In particular, OWL is already capable of expressing inconsistencies.

There are two main ways to deal with inconsistency. One is to diagnose and repair it when we encounter inconsistencies. In [17] Schlobach and Cornet propose a non-standard 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 is to simply avoid the inconsistency and to apply a non-standard reasoning method to obtain meaningful answers. In this document, we will focus on the latter, which is more suitable for web-based contexts. For example, in a typical Semantic Web setting, one would be importing ontologies from other sources, making it impossible to repair them, and the scale of the combined ontologies may be too large to make repairs effective.

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. In this document, we propose a general framework for reasoning with inconsistent ontologies. We investigate how an inconsistency reasoner can be developed for the Semantic Web. The general task of an inconsistency reasoner is: given an inconsistent ontology, return *meaningful* answers to queries. In Chapter 3 we will provide a formal definition about meaningfulness.

Reasoning with inconsistency is a well-known topic in logics and AI. Many approaches have been proposed to deal with inconsistency [4, 5, 14, 12]. The development of paraconsistent logics was initiated to challenge the ‘explosive’ problem of the standard logics. Paraconsistent logics [5] allow theories that are inconsistent but non-trivial. There are many different paraconsistent logics. Most of them are defined on a seman-

tics which allows both a letter and its negation to hold for an interpretation. Levesque's limited inference [13] allows the interpretation of a language in which a truth assignment may map both a letter l and its negation $\neg l$ to true. Extending the idea of Levesque's limited inference, Schaerf and Cadoli [16] propose S -3-entailment and S -1-entailment for approximate reasoning with tractable results. The main idea of Schaerf and Cadoli's approach is to introduce a subset S of the language, which can be used as a parameter in their framework and allows their reasoning procedure to focus on a part of the theory while the remaining part is ignored. However, how to construct and extend this subset S in specific scenarios is still an open question (the problem of finding a general optimal strategy for S is known to be intractable).

Based on Schaerf and Cadoli's S -3-entailment, Marquis and Porquet present a framework for reasoning with inconsistency by introducing a family of resource-bounded paraconsistent inference relations [14]. In Marquis and Porquet's approach, consistency is restored by removing variables from the approximation set S instead of removing some explicit beliefs from the belief base, like the standard approaches do in belief revision. Their framework enables some forms of graded paraconsistency by explicit handling of preferences over the approximation set S . In [14], Marquis and Porquet propose several policies, e.g., the linear order policy and the lexicographic policy, for the preference handling in paraconsistent reasoning. In [7], Chopra, Parikh, and Wassermann incorporate the local change of belief revision and relevance sensitivity by means of Schaerf and Cadoli's approximate reasoning method, and show how relevance can be introduced for approximate reasoning in belief revision. Both approaches, Marquis', and Chopra's, depend on syntactic selection procedures for extending the approximation set.

Our approach borrows some ideas from Schaerf and Cadoli's approximation approach, Marquis and Porquet's paraconsistent reasoning approach, and Chopra, Parikh, and Wassermann's relevance approach. However, our main idea is relatively *simple* (It is not a bad thing, isn't it?): given a selection function, which can be defined on the syntactic or semantic relevance, like those have been used in computational linguistics, we select some consistent subtheory from an inconsistent ontology. Then we apply standard reasoning on the selected subtheory to find meaningful answers. If a satisfying answer cannot be found, the relevance degree of the selection function is made less restrictive (see later sections for precise definitions of these notions) thereby extending the consistent subtheory for further reasoning.

This document is organized as follows: Chapter 2 overviews inconsistency in the Semantic Web by examining several typical examples and scenarios. Chapter 3 proposes a general framework of reasoning with inconsistent ontologies, and examines how the selection function can be developed, based on the syntactic relevance approach. Chapter 4 describes a prototype of PION. Chapter 5 discusses further work and concludes the document.

Chapter 2

Inconsistency in the Semantic Web

In the Semantic Web, inconsistencies may easily occur, sometimes even in small ontologies. Here are several scenarios which may cause inconsistencies:¹

2.1 Inconsistency by Mis-presentation of Default

When a knowledge engineer specifies an ontology statement, she/he has to check carefully that the new statement is consistent, not only with respect to existing statements, but also with respect to statements that may be added in the future, which of course may not always be known at that moment. This makes it very difficult to maintain consistency in ontology specifications. Just consider a situation in which a knowledge engineer wants to create an ontology about animals:

$bird \sqsubseteq animal$ (Birds are animals),
 $bird \sqsubseteq fly$ (Birds are flying animals).

Although the knowledge engineer may realize that ‘birds can fly’ is not generally valid, he still wants to add it if he does not find any counterexample in the current knowledge base, because flying is one of the main features of birds. An ontology about birds without talking about flying is not satisfactory.

Later on, one may want to extend the ontology with the following statements:

¹By inconsistency we mean that the set implies a contradiction, i.e., the set $\Sigma \models \perp$, given an entailment relation ‘ \models ’. It is usually called *incoherence*. We continue using the term ‘inconsistency’, because it is common practice in the ontology community.

$eagle \sqsubseteq bird$ (Eagles are birds),
 $penguin \sqsubseteq bird$ (Penguins are birds),
 $penguin \sqsubseteq \neg fly$ (Penguins are not flying animals).

The concept *penguin* in that ontology of birds is already unsatisfiable, because it implies penguins can both fly and not fly. This would lead to an inconsistent ontology. One may remove the statement ‘birds can fly’ from the existing ontology to restore consistency. However, this approach is not reliable, because of the following reasons: a) It is hard to check that the removal would not cause any significant information loss in the current ontology, b) One may not have the authority to remove statements which have been created in the current knowledge base, c) It may be difficult to know which part of the existing ontology can be removed if the knowledge base is very large. One would not blame the knowledge engineer for the creation of the statement ‘birds are flying animals’ at the beginning without considering future extensions, because it is hard for the knowledge engineer to do so.

One may argue that the current ontology languages and their counterparts in the Semantic Web cannot be used to handle this kind of problems, because it requires non-monotonic reasoning. The statement *Birds can fly* has to be specified as a default statement. The ontology language OWL cannot deal with default statements. We have to wait for an extension of OWL to accommodate non-monotonic statements. It is painful that we cannot talk about birds (that can fly) and penguins (that cannot fly) in the same ontology specification. An alternative approach is to divide the inconsistent ontology specification into multiple ontologies or modular ontologies to maintain their local consistency, like one that states ‘birds can fly’, but doesn’t talk about penguins, and another one that specifies penguins, but never mentions that ‘birds can fly’. However, the problem for this approach is still the same as other ones. Again, an ontology about birds that cannot talk about both ‘birds can fly’ and penguins is not satisfactory.

2.2 Inconsistency Caused by Polysemy

Polysemy refers to the concept of words with multiple meanings. One should have a clear understanding of all the concepts when an ontology is formally specified. Here is an example of an inconsistent ontology which is caused by polysemy:

$marriedWoman \sqsubseteq woman$	(a married woman is a woman),
$marriedWoman \sqsubseteq \neg divorcee$	(a married woman is not a divorcee),
$divorcee \sqsubseteq hadHusband \sqcap \neg hasHusband$	(a divorcee had a husband and has no husband),
$hasHusband \sqsubseteq marriedWoman$	(hasHusband means married),
$hadHusband \sqsubseteq marriedWoman$	(hadHusband means married).

In the ontology specification above, the concepts ‘divorcee’ is unsatisfiable, because of the misuse of the word ‘marriedWoman’. Therefore, one has to carefully check if there is some misunderstanding with respect to concepts that have been used in the ontology, which may become rather difficult when an ontology is large.

2.3 Inconsistency through Migration from Another Formalism

When an ontology specification is migrated from other data sources, inconsistencies may occur. As it has been found by Schlobach and Cornet in [17], the high number of unsatisfiable concepts in DL terminology for DICE is due to the fact that it has been created by migration from a frame-based terminological system.² In order to make the semantics as explicit as possible, a very restrictive translation has been chosen to highlight as many ambiguities as possible. In [17], Schlobach and Cornet show the following inconsistent ontology specification:

$brain \sqsubseteq centralNervousSystem$	(a brain is a central nervous system),
$brain \sqsubseteq bodyPart$	(a brain is a body part),
$centralNervousSystem \sqsubseteq nervousSystem$	(a central nervous system is a nervous system),
$bodyPart \sqsubseteq \neg nervousSystem$	(a body part is not a nervous system).

2.4 Inconsistency Caused by Multiple Sources

When a large ontology specification is generated from multiple sources, in particular when these sources are created by several authors, inconsistencies easily occur.

²DICE stands for ‘Diagnoses for Intensive Care Evaluation’.

In [9], Hameed, Preece and Sleeman propose approaches of ontology reconciliation, and discuss how consistency should be maintained and how inconsistency may be created from multiple sources. According to [9], there are three possibilities for ontology reconciliation: merging, aligning, or integrating. No matter whether a new ontology is generated by merging or integrating multiple sources, in both cases general consistency objectives are rather difficult to achieve.

Note that the above-mentioned categories (mis-presentation of default statements, polysemy, migration, multiple-sources) don't exclude each other. When we examine an inconsistent ontology which is generated from multiple sources, we may find that it contains several cases of polysemy, or some other inconsistency. The list above is also not exhaustive. There are many other cases that may cause the inconsistency, like inconsistency caused by ambiguities, inconsistency caused by lacking global checking, etc. We do not discuss a complete list in this document, but we have just aimed to show the urgency of the problem of reasoning with inconsistent ontologies.

Chapter 3

Framework of Reasoning with Inconsistent Ontologies

In this chapter we present our framework for reasoning with inconsistent ontologies. First we will introduce some definitions and terminology for inconsistency reasoners, for instance what do we mean with a ‘meaningful’ answers to a query. We continue with identifying when to use the inconsistency reasoner. Finally a concrete strategy for inconsistency processing and a selection function for an specific inconsistency reasoner is discussed, and illustrated with two examples.

3.1 Formal Definitions

This document proposes a general framework for reasoning with inconsistent ontologies. Therefore, we do not restrict ontology specifications to a particular language (although OWL and its description logic are the languages we have in mind). In general, an ontology language can be considered to be a set that is generated by a set of syntactic rules. Thus, we can consider an ontology specification as a formula set. We use a non-classical entailment for inconsistency reasoning. In the following, we use \models to denote the classical entailment, and use \approx to denote some non-standard inference relation, which may be parameterized to remove ambiguities.

In general, an ontology query Σ can be expressed as ‘ $\Sigma \approx \phi?$ ’, where ϕ is a formula. There are two standard answers to a query, either ‘yes’ ($\Sigma \approx \phi$), or ‘no’ ($\Sigma \not\approx \phi$).¹

For an inconsistency reasoner it is expected that is able to return meaningful answers to queries, given an inconsistent ontology. In the following we propose several formal definitions for inconsistency reasoners.

¹In the next subsection, we will argue that it would be more suitable for an inconsistency reasoner to extend the query answers to three possible answers: ‘accepted’ ($\Sigma \approx \phi$ and $\Sigma \not\approx \neg\phi$), ‘rejected’ ($\Sigma \approx \neg\phi$ and $\Sigma \not\approx \phi$), or ‘undetermined’ ($\Sigma \not\approx \phi$ and $\Sigma \not\approx \neg\phi$).

Soundness: In the case of a consistent ontology Σ , classical reasoning is sound, i.e., a formula ϕ deduced from Σ holds in every model of Σ . This definition is not preferable for an inconsistent ontology Σ as every formula follows from it using classical entailment. However, often only a small part of Σ has been incorrectly constructed or modelled, while the remainder of Σ is sound. This leads us to the following: an inconsistency reasoner should be considered sound if the formulas that follow from an inconsistent theory Σ follow from a consistent subtheory of Σ using classical reasoning. Therefore, we propose the following definition of soundness. An inconsistency reasoner \approx is sound if the following condition holds:

$$\Sigma \approx \phi \Rightarrow (\exists \Sigma' \subseteq \Sigma)(\Sigma' \not\equiv \perp \text{ and } \Sigma' \models \phi).$$

In other words, the \approx -consequences must be justifiable on the basis of a consistent subset of the theory. Note however, that in the previous definition the implication should *not hold* in the opposite direction. If the implication would also hold in the opposite direction it would lead to an inconsistency reasoner, which returns inconsistent answers. For example if $\{a, \neg a\} \subseteq \Sigma$, then the inconsistency reasoner would return that both a and $\neg a$ hold given Σ , which is something we would like to prevent. Hence, the inconsistency reasoner should not return answers that follow from *any* consistent subset of Σ , but from *specifically chosen* subsets of Σ . This ‘specific selection of consistent subsets’ is part of the inconsistent reasoners strategy and will be discussed in more detail in Section 3.2.2.

Meaningfulness: An answer given by an inconsistency reasoner is meaningful iff it is consistent and sound. Namely, it requires not only the soundness condition, but also the following condition:

$$\Sigma \approx \phi \Rightarrow \Sigma \not\approx \neg\phi.$$

Inconsistency reasoning is said to be meaningful iff all of the answers are meaningful.

Local Completeness: Because of inconsistencies, classical completeness is impossible. We suggest the notion of local completeness: inconsistency reasoning is locally complete with respect to a consistent subtheory Σ' iff for any formula ϕ , the following condition holds:

$$\Sigma' \models \phi \Rightarrow \Sigma \approx \phi.$$

Since the condition can be represented as:

$$\Sigma \not\approx \phi \Rightarrow \Sigma' \not\models \phi,$$

local completeness can be considered as a complement to the soundness property. An answer to a query $\Sigma \approx \phi?$ is said to be locally complete with respect to a consistent set Σ' iff the following condition holds:

$$\Sigma' \models \phi \Rightarrow \Sigma \approx \phi.$$

Maximality: An inconsistency reasoner is maximal iff there is a maximal consistent subtheory such that its consequence set is the same as the consequence set of the inconsistency reasoner:

$$\exists(\Sigma' \subseteq \Sigma)((\Sigma' \not\models \perp) \wedge (\forall \Sigma'' \supset \Sigma' \wedge \Sigma'' \subseteq \Sigma)(\Sigma'' \models \perp) \wedge \forall \phi(\Sigma' \models \phi \Leftrightarrow \Sigma \approx \phi)).$$

We use the same condition to define the maximality for an answer, like we do for local completeness.

Local Soundness: An answer to a query ‘ $\Sigma \approx \phi?$ ’ is said to be locally sound with respect to a consistent set $\Sigma' \subseteq \Sigma$, iff the following condition holds:

$$\Sigma \approx \phi \Rightarrow \Sigma' \models \phi.$$

Namely, for any positive answer, it should be implied by the given consistent subtheory Σ' under the standard entailment.

From the definitions given above it follows that local soundness implies soundness and meaningfulness. Moreover, it follows that maximal completeness implies local completeness. Given a query, there might exist more than one maximal consistent subset. The same also holds for local completeness. Such different maximally consistent subsets may give different \approx -consequences for a given query ϕ . Therefore, arbitrary (maximal) consistent subsets may not be very useful for the evaluation of a query by some inconsistency reasoner. The consistent subsets should be chosen on structural or semantical grounds indicating the relevance of the chosen subset with respect to some query. Functions that will select specific subsets of an ontology Σ will be discussed in more detail in Section 3.3.

3.2 Algorithms

3.2.1 Inconsistency Detection

With classical reasoning, a query ϕ given an ontology Σ can be expressed as an evaluation of the consequence relation $\Sigma \models \phi$. There are only two answers to that query: either ‘yes’ ($\Sigma \models \phi$), or ‘no’ ($\Sigma \not\models \phi$). A ‘yes’ answer means that ϕ is a logical conclusion of Σ . A ‘no’ answer, however, means that ϕ cannot be deduced from Σ , because we usually don’t adopt the closed world assumption when using an ontology. Hence, a ‘no’ answer does not imply that the negation of ϕ holds given an ontology Σ . For reasoning with inconsistent ontologies, it is more suitable to use Belnap’s four valued logic [3] to distinguish the following four epistemic states of the answers:

Definition 1

- *Over-determined:* $\Sigma \models \phi$ and $\Sigma \models \neg\phi$.
- *Accepted:* $\Sigma \models \phi$ and $\Sigma \not\models \neg\phi$.
- *Rejected:* $\Sigma \not\models \phi$ and $\Sigma \models \neg\phi$.
- *Undetermined:* $\Sigma \not\models \phi$ and $\Sigma \not\models \neg\phi$.

To make sure reasoning is reliable when it is unclear if an ontology is consistent or not, we can use the decision tree that is depicted in Figure 3.1. For a query ϕ we test both the consequences $\Sigma \models \phi$ and $\Sigma \models \neg\phi$ using classical reasoning. In case different answers are obtained, i.e., both ‘yes’ and ‘no’, the ontology Σ must be consistent and the answer to $\Sigma \models \phi$ can be returned. In case of two answers that are the same the ontology is either incomplete, i.e., when both answers are ‘no’, or the ontology is inconsistent, i.e., when both answers are ‘yes’. When the ontology turns out to be incomplete, either an ‘undetermined’ answer can be returned or additional information I can be gathered to answer the query $\Sigma \cup I \models \phi$, but this falls outside the scope of this document. When the ontology turns out to be inconsistent some inconsistency reasoner can be called upon to answer the query $\Sigma \models \phi$.

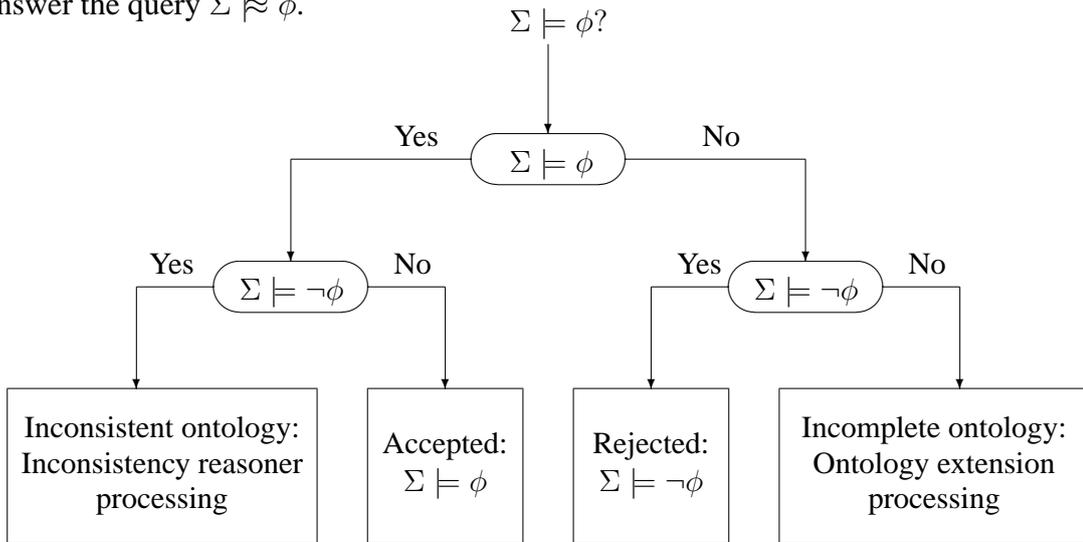


Figure 3.1: Decision tree for obtaining reliable reasoning with an inconsistent ontology.

3.2.2 Inconsistency Reasoning

An inconsistency reasoner uses a selection function to determine which consistent subsets of an inconsistent ontology should be considered in its reasoning process. The general framework of Figure 3.1 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 [7], or based on semantic relevance like for example in computational linguistics as in Wordnet [6].

In this document we will consider only monotonic selection functions:

Definition 2 *A selection function f is called monotonic if the consistent subsets that it selects S_1, S_2, S_3, \dots monotonically increase or decrease, e.g., $S_1 \subseteq S_2 \subseteq S_3 \subseteq \dots$.*

In particular we will restrict ourselves to monotonically increasing selection functions. Monotonic selection functions have the advantage that they do not have to return *all* subsets for consideration at the same time. If a query $\Sigma \approx \phi$ can be answered after considering some consistent subset of the ontology Σ given by the selection function, other subsets don't have to be considered any more, because *they will not change the answer of the inconsistency reasoner*.

Because of this property, in this document we always use an iterative approach. An inconsistency reasoner that uses a monotone selection function will be called an inconsistency reasoner that uses a *linear extension strategy*. A linear extension strategy is carried out as shown in Figure 3.2. Given a query $\Sigma \approx \phi$, the initial consistent subset Σ' is set to be equal to the empty set \emptyset . Then the selection function is called to return a consistent subset Σ'' that extends Σ' , i.e., $\Sigma' \subset \Sigma'' \subseteq \Sigma$. If the set Σ'' does not exist, the inconsistency reasoner returns the answer 'undetermined' 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 inconsistency reasoner returns the 'accepted' answer $\Sigma'' \models \phi$. If the answer is 'no', the inconsistency reasoner further checks the negation of the query $\Sigma'' \models \neg\phi$. If the answer is 'yes', the inconsistency reasoner returns the 'rejected' answer $\Sigma \approx \neg\phi$, otherwise 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 the following properties hold:

Proposition 3.2.1 (Linear Extension) *An inconsistency reasoner using a linear extension strategy satisfies the following properties:*

- *never over-determined,*
- *may be undetermined,*
- *always sound,*

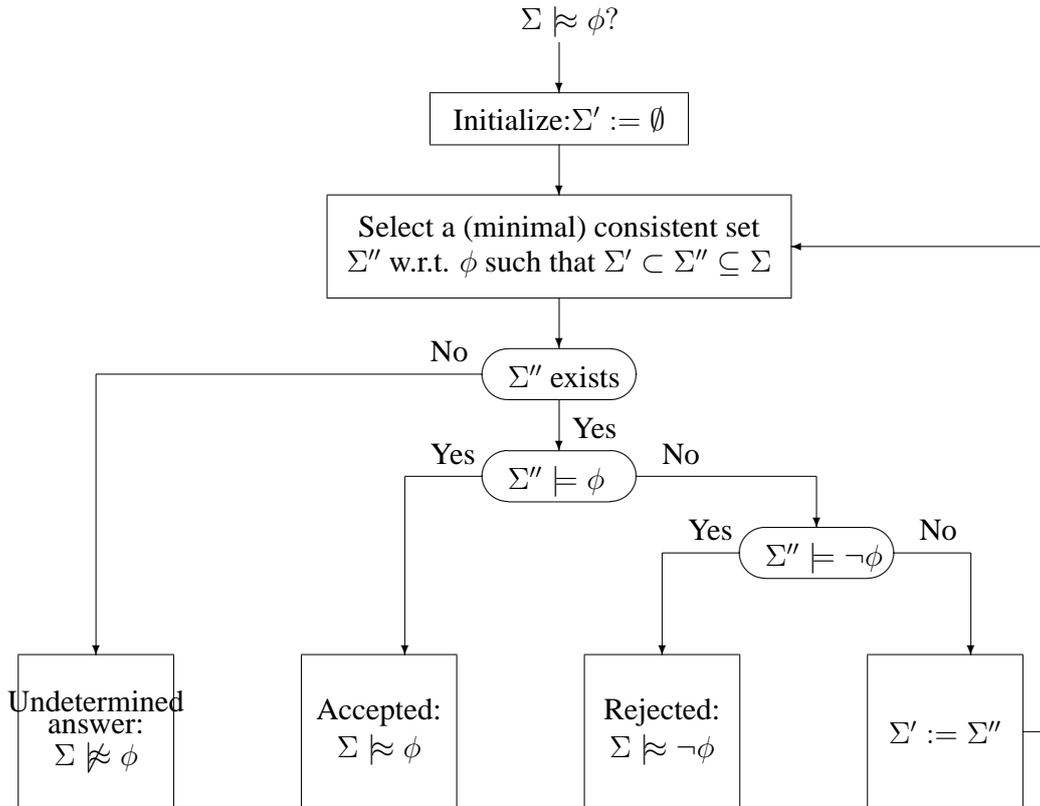


Figure 3.2: An inconsistency reasoner using the Linear Extension strategy.

- *always meaningful,*
- *always locally complete,*
- *may not be maximal,*
- *always locally sound.*

Therefore, an inconsistency 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' if the selection function always starts with the consistent working set Σ' . Unfortunately it may not be maximal. We call this strategy a *linear* one, because the selection function only follows one possible ‘extension chain’ for creating consistent subsets. The advantages of the linear strategy is that the reasoner can always focus on the current working set Σ' . The reasoner doesn’t need to keep track of the extension chain. The 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.

3.3 Selection Function

As we have pointed out in Section 3, the definition of the selection function should be independent of the general procedure of the inconsistency processing (i.e., strategy). Further research will focus on a formal development of selection functions. However, we would like to point out that there exist several alternatives which can be used for an inconsistency reasoner.

In [7], Chopra, Parikh, and Wassermann propose 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. We will exploit their relevance measure as selection function and illustrate them on two examples.

Definition 3 (Direct relevance and k -relevance [7]) *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 $p_1, p_2, \dots, p_k \in \mathcal{L}$ such that:*

- p, p_1 are directly relevant;
- p_i, p_{i+1} are directly relevant, $i = 1, \dots, k - 1$;
- p_k, q are directly relevant.

The notions of relevance are based on propositional logics. However, ontology languages are usually written in some subset of first order logic. It would not be too difficult to extend the ideas of relevance to those first-order logic-based languages by considering an atomic formula in first-order logic as a primitive proposition in propositional logic. In a sequel report, we investigate this extension of relevance definitions formally. However, in this document, we just adapt the idea informally.

The following definition specialises the general definition of relevance for the case where ϕ is a formula in an ontology.

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. Two formula ϕ, ψ are directly relevant iff there is a common name which appears both in formula ϕ and formula ψ , i.e., $I(\phi) \cap I(\psi) \neq \emptyset \vee C(\phi) \cap C(\psi) \neq \emptyset \vee R(\phi) \cap R(\psi) \neq \emptyset$. 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 specialise the notion of k -relevance.

In inconsistency reasoning we can use syntactic relevance to define a selection function to extend the query ' $\Sigma \models \phi?$ ' as follows: we start with the query formula ϕ as a starting point for the selection based on syntactic relevance. First we select 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. 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. The selection procedure can be described as two working sets Σ' and Σ'' as shown in the linear extension algorithm of Figure 3.2. At the beginning, the first working set Σ' is defined with respect to the query ϕ and the formula set Σ as follows:

$$\Sigma' = \{\psi \in \Sigma \mid \phi \text{ and } \psi \text{ are directly relevant}\}.$$

The second working set is based on the first working set Σ' as follows:

$$\Sigma'' = \{\psi \in \Sigma \mid \psi \text{ is relevant to } \Sigma'\}.$$

Example bird ontology: Consider the bird ontology as defined in Section 2. We add a fact $penguin(tweety)$ (Tweety is a penguin) into the A-box of the ontology. Note that we need that fact to state that the Penguin exists, namely, Penguin is not a bottom concept. Let Σ be the formula set of the bird ontology. Then, $\Sigma_1 = \Sigma \cup \{penguin(tweety)\}$ is an inconsistent formula set. For the query ' $\Sigma \approx fly(tweety)?$ ' (can Tweety fly?), at the beginning, the selection function would select the formula set $\Sigma'' = \{bird \sqsubseteq fly, penguin \sqsubseteq \neg fly, penguin(tweety)\}$, based on the direction relevance. We have

$$\Sigma'' \models \neg fly(tweety).$$

Therefore, the inconsistency reasoner returns a negative answer to the query. Namely, we get the intended answer:

$$\Sigma_1 \approx \neg fly(tweety).$$

The same result would also be valid for the query ' $\neg fly(tweety)?$ ' (Can Tweety not fly?). For the queries on other birds, i.e., not about penguins, the reasoning process would always give the intended results, because the statement $penguin \sqsubseteq bird$ would not be involved. Therefore, it would not lead to an inconsistency.

Example brain ontology: Consider the brain ontology example. Similarly, we add a fact $brain(a)$ into the A-box to state that the concept 'brain' is not empty. For the query $nervousSystem(a)?$ (Is the object 'a' an element of a nervous system?), based on the direct relevance to the formula $nervousSystem(a)$, the selection function will select the following formula set at the beginning:

$$\Sigma_3 = \{centralNervousSystem \sqsubseteq nervousSystem, \\ bodyPart \sqsubseteq \neg nervousSystem, brain(a)\}.$$

However, the query is undetermined on this selected set. The selection function will extend the set with more formulae, of lower relevance (in this case $k=2$). Unfortunately,

this would lead to all formulas in the example being selected, which is inconsistent, making the query over-determined. To solve this problem, the selection function should select a consistent set larger than Σ_3 , while still smaller than the entire theory. We call the procedure *over-determined processing*. A possibility is to include not all $k = 2$ -relevant formula, but only make the $k = 2$ extension w.r.t. to some of the atoms appearing in Σ_3 . In particular, the following set (Σ_4) is based on the extension of the atom *centralNervousSystem*

$$\Sigma_4 = \{brain \sqsubseteq centralNervousSystem\} \cup \Sigma_3.$$

On this set, the answer to the query is positive, therefore, the reasoning process can stop and return the accepted answer *nervousSystem(a)*.

The notions of direct relevance and k -relevance are a syntactic approach. As we have shown above, the syntactic relevance approach can give intuitive results for most cases, because we can consider the formulas that have been stated in the ontologies to be explicit beliefs/knowledge, and the formulas that can be derived from the existing formula sets to be implicit beliefs/knowledge. We can count the relevance on the explicit knowledge, instead of the implicit knowledge. More cases are needed to evaluate syntactic relevance approaches for reasoning with inconsistent ontologies. The semantic relevance approaches, which measure the relevance on a semantic level instead of a syntactic level, can be used as alternatives of the selection functions for reasoning with inconsistent ontologies. We will investigate this in the near future.

Chapter 4

Prototype of a Reasoner with Inconsistent Ontologies

4.1 Introduction

We have implemented the prototype of PION by using SWI-Prolog.¹ SWI-Prolog is a free software Prolog compiler. Being free, small and mostly standard compliant, SWI-Prolog has become very popular for education and research. We are using a logic programming language like Prolog as the tool for the implementation of the prototype, because the logic programming language is convenient and powerful for symbolic processing and list processing. For the implementation of PION, we have developed XDIG, an extended DIG description logic interface package for Prolog[11], which provides a general infrastructure to build (hybrid) reasoning frameworks. A PION serves as a DL reasoner via its own DIG interface. It is designed to be a simple API for a general reasoner with inconsistent ontologies. It supports DIG requests from other ontology applications or other ontology and metadata management systems. Thus, the implementation of PION will be independent from those particular applications or systems.

4.2 Architecture

The architecture of a PION is designed as an extension of the XDIG framework, and is shown in Figure 4.1. A PION consists of the following components:

- **DIG Server:** The standard XDIG server acts as PION's DIG server, which deals with requests from other ontology applications. It not only supports standard DIG requests, like 'tell' and 'ask', but also provides additional reasoning facilities, like the identification of the reasoner or change of the selected selection functions.

¹<http://www.swi-prolog.org>

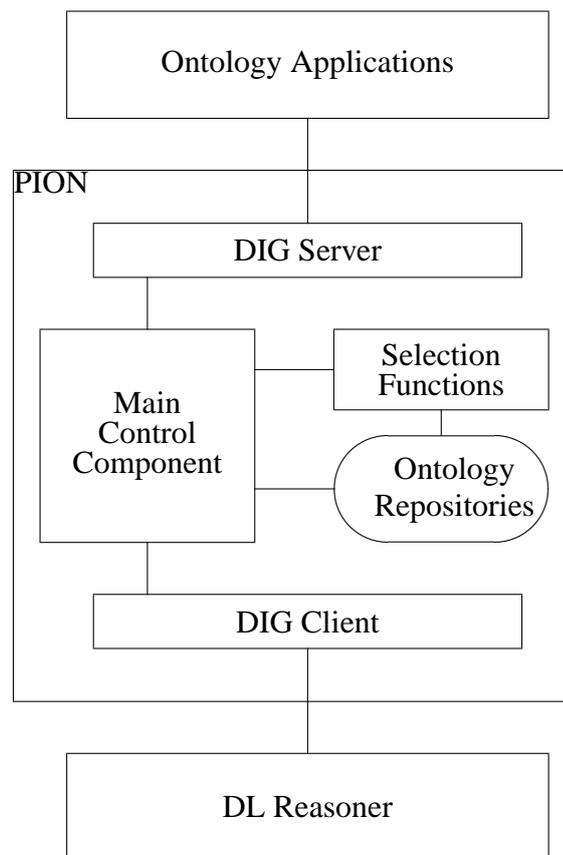


Figure 4.1: Architecture of PION

- **Main Control Component:** The main control component performs the main processing, like query analysis, query pre-processing, and the extension strategy, by calling the selection function and interacting with the ontology repositories.
- **Selection-Functions:** The selection function component is an enhanced component to XDIG, it defines the selection functions that may be used in the reasoning process.
- **DIG Client:** PION's DIG client is the standard DIG client, which calls external DL reasoners that support the DIG interface to obtain the standard DL reasoning capabilities.
- **Ontology Repositories:** The ontology repositories are used to store ontology statements, provided by external ontology applications.

4.3 Implementation

PION is implemented by means of the XDIG package. XDIG provides a general infrastructure for the construction of hybrid reasoning frameworks and facilitates the specification of user defined processing scenarios, to be specified in a 'my_dig_server_processing' predicate. The predicate 'my_dig_server_processing' serves as the main entry point of the server; it handles both 'tell' and 'ask' requests. For a TELL request, PION stores the data into its own local ontology repositories; for ASK requests, PION posts the stored data in the repositories, after which it queries the external DL reasoner, similar to the accumulation strategy as described in [11].

If the answers from the external reasoner do not report an error condition, the current ontology is consistent and PION uses the answers from the external reasoner as its own answer to the applications. Inconsistencies usually occur for instance queries, like 'Is Tweety a flying animal?'. Satisfiability and other queries would not result in an inconsistency error. The current prototype version of PION only deals with instance inconsistencies.

In case the current ontology is inconsistent, PION will start an inconsistency processing scenario by launching a search strategy, which uses one of the selection functions in the system to find a relevant and consistent subset of the current ontology and pass the selected subset to the external standard DL reasoner.

4.4 Functionalities

The current version of the PION prototype (version 1.0) has the following characteristics:

- **Strategy:** linear extension strategy.

- **Selection Function:** a syntactic relevance-based selection function.
- **Over-determined processing:** the selection function returns the first maximally coherent subset of the over-determined set.
- **Ontology and Query Languages:** the internal data format is the DIG format, however, PION supports the DIG format as well as the OWL format for TELL requests and the DIG format for ASK requests. Ontological data in the OWL format is translated automatically by the XDIG component 'owl2dig'².

4.5 Installation

1. **Download:** The PION package (version 1.0) is available from the SEKT website or the VUA SEKT website:

`http://wasp.cs.vu.nl/sekt/download/sekt341.zip`

Unzip the PION package 'sekt341.zip' into a directory. We will call the directory *PION_ROOT*.

2. **Installation of SWI-Prolog:** PION requires that SWI-Prolog (version 5.4.4 or higher) has been installed on your computers. It can be downloaded from the SWI-Prolog website:

`http://www.swi-prolog.org`

Install SWI-Prolog into a directory. We will call that directory *SWIPROLOG_ROOT*.

3. **Installation of XDIG:** You can find the zip file 'diglibrary.zip', the XDIG package which includes the PION libraries in the directory *PION_ROOT*. Unzip the file 'diglibrary.zip' into the SWI-Prolog library directory, i.e., *SWIPROLOG_ROOT/library*.
4. **Installation of Racer:** PION requires Racer (version 1.7.14 or higher) as the external DL reasoner. Other DL reasoners may work for PION if they support the DIG DL interface, however, they have not yet been tested. Racer can be downloaded from the Racer website:

`http://www.sts.tu-harburg.de/~r.f.moeller/racer/download.html`

²However, note that the component 'owl2dig' is still under development. The complete specification of OWL DL is not yet supported.

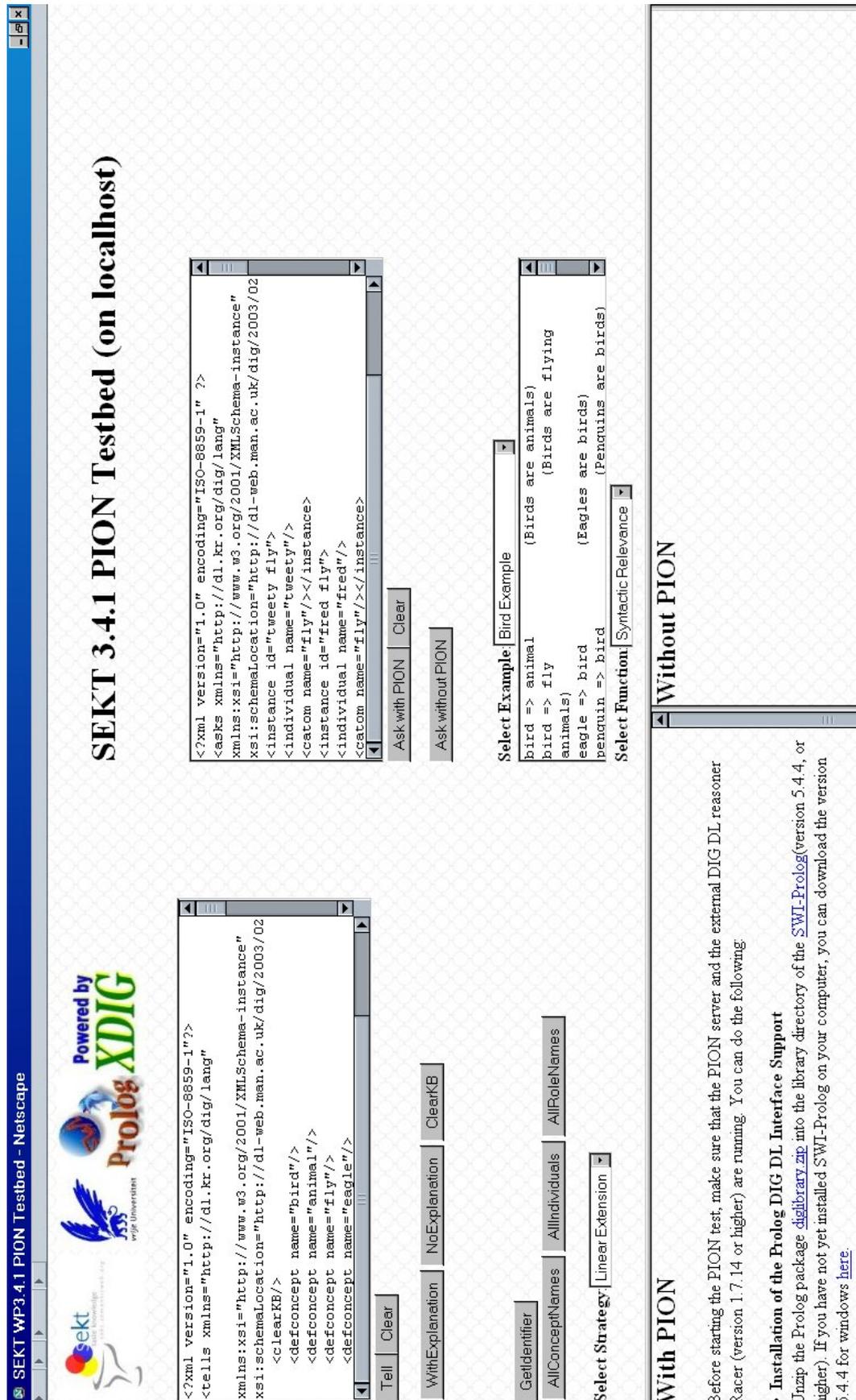


Figure 4.2: PION testbed.

4.6 PION Testbed

The PION testbed 'piontest2.htm' is a PION client with a graphical interface, which is designed as a webpage. Therefore it can be launched from a web browser which supports Javascript. A screenshot of the PION testbed, is shown in Figure 4.2.

The current version of the PION testbed supports tests for which both the PION server (with default port: 8001) and the external DL reasoner (with the default port: 8000) are running on the localhost. For a PION server which runs on a remote host, change the host and port data in the 'pionmain2.htm' file. Namely, replace the URL 'http://127.0.0.1:8001' with another valid URL.

The PION testbed allows for the comparison of query results with PION and without PION. Query results without PION are realized by a simply-forward server 'dig_server_simple8002.pl', which deals with standard DIG requests and OWL support. Before starting the PION test, make sure that the PION server, the simply-forward server and the external DL reasoner (i.e. Racer) are running at the host with the correct ports. The default port of the PION server port is '8001'. The default port of the external DL reasoner is '8000'. The default port of the simply-forward server is '8002'. The default hostname of the PION server and the external DL reasoner is 'localhost'.

1. **Launch Racer:** Racer can be launched by the following command: `racer -http 8000` Alternatively, click on the file 'racer8000.bat' if the PION download package includes the Racer reasoner.
2. **Launch PION server:** click on the file 'pion_server.pl' in the PION directory.
3. **Launch the simply-forward server:** click on the file 'dig_server_simple8002.pl' on the PION directory.

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

As mentioned in Section 4.3, the current version of PION deals only with instance inconsistencies. Answers from the external DL reasoner are either 'true', 'false' or 'error' for instance queries. If the ontology is consistent, PION will return the answers directly to the user. If the ontology is inconsistent, PION will start the inconsistency processing. The answers from the inconsistency processing are either 'accepted', 'rejected' or 'undetermined'.

The PION testbed offers several typical examples for PION tests, which include the bird example, the brain example, the married woman example, which are discussed in Chapter 2. These examples are in the DIG format. The PION testbed also offers the mad cow example and a variant example in OWL.

SEKT 3.4.1 PION Testbed (on localhost)



```

<?xml version="1.0" encoding="ISO-8859-1"?>
<tells xmlns="http://dl.kr.org/dig/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://dl-
web.man.ac.uk/dig/2003/02/dig.xsd">
  <clearKB/>
  <defconcept name="bird"/>
  <defconcept name="animal"/>
  <defconcept name="fly"/>
  <defconcept name="eagle"/>
</tells>

```

Tell Clear

WithExplanation NoExplanation ClearKB

GetIdentifier

AllConceptNames AllIndividuals AllRoleNames

Select Strategy: Linear Extension

```

<?xml version="1.0" encoding="UTF-8" ?>
- <response>
  <rejected id="tweety fly" />
  <accepted id="fred fly" />
  <accepted id="tweety not fly" />
  <rejected id="fred not fly" />
  <accepted id="tweety bird" />
  <accepted id="fred bird" />
  <true id="satisfiable fly" />
  <false id="satisfiable penguin" />
</response>

```

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<asks xmlns="http://dl.kr.org/dig/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://dl-
web.man.ac.uk/dig/2003/02/dig.xsd">
  <instance id="tweety fly">
  <individual name="tweety"/>
  <atom name="fly"/></instance>
  <instance id="fred fly">
  <individual name="fred"/>
</asks>

```

Ask with PION Clear

Ask without PION

Select Example: Bird Example

```

bird => animal (Birds are animals)
bird => fly (Birds are flying
animals)
eagle => bird (Eagles are birds)

```

Select Function: Syntactic Relevance

```

<?xml version="1.0" encoding="UTF-8" ?>
- <responses xmlns="http://dl.kr.org/dig/2003/02/lang"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://dl.kr.org/dig/2003/02/dig.xsd">
  <error id="tweety fly" message="ABox DEFAULT is incoherent." />
  <error id="fred fly" message="ABox DEFAULT is incoherent." />
  <true id="satisfiable fly" />
  <error id="tweety not fly" message="ABox DEFAULT is incoherent." />
  <error id="fred not fly" message="ABox DEFAULT is incoherent." />
  <false id="satisfiable penguin" />
  <error id="tweety bird" message="ABox DEFAULT is incoherent." />
  <error id="fred bird" message="ABox DEFAULT is incoherent." />
</responses>

```

Figure 4.3: Bird Example on PION testbed

PION supports the comparison of query results with PION and query results without PION, which are displayed in the left-bottom text area and the right-bottom text area respectively.

A screenshot of the PION testbed, displaying the answers with PION and without PION for the bird example, is shown in Figure 4.3.

The explanation of an accepted / rejected query answer can be obtained by clicking on the 'withExplanation' button before performing the ASK request. An explanation of the answer is the selected consistent set from which the result is derived. A query answer with the explanation of the bird example is shown as follows:

```
<response>
  <rejected id="tweety fly">
    <because>
      <impliesc>
        <catom name="bird"/>
        <catom name="fly"/>
      </impliesc>
      <impliesc>
        <catom name="penguin"/>
        <not>
          <catom name="fly"/>
        </not>
      </impliesc>
      <instanceof>
        <individual name="tweety"/>
        <catom name="penguin"/>
      </instanceof>
    </because>
  </rejected>
</response>
```

Chapter 5

Discussion and Conclusions

In this document, we proposed a general framework for reasoning with inconsistent ontologies. We have introduced several formal definitions, such as soundness, meaningfulness, local completeness, and maximal completeness for inconsistency reasoning. We have proposed a pre-processing algorithm and an inconsistency reasoning strategy, based on a linear extension strategy, and also have shown that a linear extension strategy is useful for creating meaningful and sound answers to queries, although they may be undetermined and not always maximal.

We have presented a prototype of PION, and discussed the architecture and implementation issues of a PION system, which is based on the general framework that has been proposed in this document. A PION testbed has been designed and implemented to offer a tool for the data tests.

We have discussed how the selection function can be developed by means of a syntactic relevance measure, and have shown several examples of how the syntactic relevance approach can be used to obtain intuitive reasoning results in most cases. However, the syntactic relevance based selection function does not always provide the intended results. For example, for the mad cow example which is shown in the PION testbed, the query 'is the mad cow moo2 a vegetarian' is accepted by PION. This counter-intuitive answer results from the weakness of the syntactic relevance-based selection function, because it always prefers a shorter relevance path. In the mad cow example, the path 'mad cow - cow - vegetarian' is shorter than the path 'mad cow - eat brain - eat bodypart - sheep are animals - eat animal - NOT vegetarian'. Therefore, the syntactic relevance-based selection function finds a consistent sub-theory by simply ignoring the fact 'sheep are animals'.

There are several alternative approaches to solve this kind of problems. One is to introduce the locality requirement. Namely, the selection function starts with a certain sub-theory which must always be selected. For example, the statement 'sheep are animals' can be considered to be a knowledge statement which cannot be ignored. Another approach is to add a shortcut path, like the path 'mad cow - eat animal - NOT vegetarian' to achieve the relevance balance between the concepts 'vegetarian' and 'NOT vegetarian',

as shown in the second mad cow example of the PION testbed. The latter approach can be achieved automatically by accommodation of the semantic relevance from the user queries. The hypothesis is that both concepts appear in a query more frequently, when they are semantically more relevant. Therefore, from a semantical point of view, we can add a relevance shortcut path between strong relevant concepts.

In future work, we will investigate formal properties of selection functions as well as different approaches to selection functions (e.g. semantic-relevance based). We will also investigate different extension strategies as alternatives to the linear extension strategy in combination with different selection functions (also instance semantic relevance based), and evaluate their performance characteristics.

Bibliography

- [1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, Peter Patel-Schneider, (eds.), *The Description Logic Handbook Theory, Implementation and Applications*, Cambridge University Press, Cambridge, UK, 2003.
- [2] Sean Bechhofer, Ralf Møller, and Peter Crowther. The DIG Description Logic Interface. International Workshop on Description Logics, Rome, September 2003.
- [3] N. Belnap, A useful four-valued logic, in: *Modern Uses of Multiple-Valued Logic*, Reidel, Dordrecht, 1977, pp.8-37.
- [4] Salem Benferhat, and Laurent Garcia, Handling locally stratified inconsistent knowledge bases, *Studia Logica*, 70: 77-104, Kluwer Academic Publishers, 2002.
- [5] Jean-Yves Beziau, What is paraconsistent logic, in: Batens D., Mortensen C., Priest G. and Van Bendegem J.P. (eds). *Frontiers of paraconsistent logic*. Research Studies Press: Baldock, 2000, 95-111.
- [6] Alexander Budanitsky and Graeme Hirst, Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. Workshop on WordNet and Other Lexical Resources, 2nd meeting of the North American Chapter of the Association for Computational Linguistics. Pittsburgh, PA. 2001.
- [7] Samir Chopra, Rohit Parikh, and Renata Wassermann, Approximate belief revision-preliminary report, *Journal of IGPL*, Oxford University Press, 2000.
- [8] Volker Haarslev, and Ralf Møller, Description of the RACER System and its Applications, Proceedings of the International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August 2001, pp. 132-141.
- [9] A. Hameed, A. Preece and D. Sleeman, Ontology Reconciliation, in: S. Staab and R. Studer (eds), *Handbook on Ontologies in Information Systems*, Springer Verlag, 231-250, 2003.
- [10] Zhisheng Huang, Frank van Harmelen, Annette ten Teije, Perry Groot, Cees Visser, Reasoning with Inconsistent Ontologies: a general framework, SEKT Report D3.4.1.1, 2004.

- [11] Zhisheng Huang and Cees Visser, Extended DIG Description Logic Interface Support for Prolog, SEKT Report D3.4.1.2, 2004.
- [12] Jerome Lang, and Pierre Marquis, Removing inconsistencies in assumption-based theories through knowledge-gathering actions, *Studia Logica*, 67: 179-214, 2001.
- [13] H. J. Levesque, A knowledge-level account of abduction, *Proceedings of IJCAI'89*, 1061-1067, 1989.
- [14] Pierre Marquis and Nadege Porquet, Resource-bounded paraconsistent inference, *Annals of Mathematics and Artificial Intelligence* 39: 349-384, 2003.
- [15] Russell, S., and Norvig, P., *Artificial Intelligence*, Prentice Hall, 1995.
- [16] Marco Schaerf and Marco Cadoli, Tractable reasoning via approximation. *Artificial Intelligence*, 74:249-310, 1995.
- [17] Stefan Schlobach, and Ronald Cornet, Non-standard reasoning services for the debugging of description logic terminologies, *Proceedings of IJCAI 2003*, 2003.