Justifications for Inconsistency under Fixed-Domain Semantics

MASTER THESIS

Submitted on: 13. September 2017
by: Satyadharma Tirtarasa
Student-ID 4108227

Supervisor: Prof. Sebastian Rudolph
Faculty of Computer Science, Institute of Artificial Intelligence
Professorship Computational Logic

Advisor: M.Sc. Lukas Schweizer
Faculty of Computer Science, Institute of Artificial Intelligence

TECHNISCHE UNIVERSITÄT DRESDEN
Declaration

I hereby declare, that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole, or in part to obtain a degree or any other qualification neither in this nor in any other university. This thesis is my own work, only with the help of the referenced literature and under the careful supervision of my thesis supervisor. No other resources apart from the references and auxiliary means indicated in the bibliography were used in the development of the presented work.

(Place, Date) Satyadharma Tirtarasa
Wundstraße 7, Dresden
First of all, I would like to express my deepest gratitude to my supervisor Prof. Sebastian Rudolph and my advisor M.Sc. Lukas Schweizer. Their guidance and support from the beginning until the completion of this thesis are invaluable. All of their feedbacks and suggestions made this work better and better. This experience has taught me a lot as a person, not to mention all of the knowledge that I learned along the way.

I would like to thank the Joint Commission of the European Masters Program in Computational Logic, especially Prof. Stefen Hlldobler as the EMCL coordinator and Prof. Sergio Tessaris as the local coordinator at unibz Bolzano. I had the greatest time of studying at both TU Dresden and unibz. The organization helped students in all the formalities seamlessly.

My deepest thanks to my family who thousands of kilometers apart from me. Their encouragements and supports let me to keep walking. I would also like to thank all of my friends both in Europe and Indonesia. With them, I enjoy every moment of my life. Lastly, I want to thank everyone who will make this list endless.

"Isn't it a pleasure to study and practice what you have learned?" - Confucius
Abstract

Description logics (DLs) are arguably one of the important knowledge representation languages today. The number of the tools available are getting larger each year. The growth itself is helped by the fact that DLs are the underlying logic of the web ontology language (OWL). However, building a knowledge base is prone to errors. It is not uncommon that a knowledge base designer finds an unwanted entailment. The justification framework gives a solution to tackle this problem by giving minimal subsets that are responsible for an entailment. On the other hand, we have a non-standard fixed-domain semantics that gives a complexity advantage over expressive DLs and more intuitive models for certain cases. In this work, we look into how the justification framework and the fixed-domain semantics entangle.

We provide two approaches, glass-box and black-box, for finding justifications for inconsistency under the fixed-domain semantics. In the black-box approach we use the combination of existing justification framework tools with a knowledge base axiomatization for fixed-domain semantics. On the other hand, we define a translation from a SROIQ knowledge base into an answer set programming (ASP) program in the glass-box approach. Answer sets of the program coincide with justifications for inconsistency of the knowledge base. We also prove that this encoding is sound and complete.

The result shows that there is no clear winner between two approaches. Each approach has cases where they perform better than the other. Although it is not feasible getting all justifications for big problem instances, getting some of them is achievable. Thus, we are sure that this work gives a starting point and some steps forwards providing the justification framework for the fixed-domain semantics. Practically, we equipped the users with a debugging tool in modelling their knowledge base under the fixed domain semantics.
1 Introduction

This chapter provides an introduction, which gives a glance what you can expect in the thesis. We start by portraying our motivation in Section 1.1. Then, the concrete problem statements are presented in Section 1.2. A brief summary of the others chapters can be found in Section 1.3.

1.1 Motivation

The Web Ontology Language [W3C09] (OWL) is one of the important knowledge representation formalisms used over various domains. The OWL is a family of languages that is used as a standard by the World Wide Web Consortium (W3C) for exchanging knowledge to support the Semantic Web. The OWL popularity is assisted by the fact that it is built on Description Logics (DLs) as a strong foundation. DLs provide more expressive power than propositional logic, but are still decidable, unlike first order logic. Due to this fact, the OWL does not only provide a modeling syntax, but a capability of inferring implicit knowledge lies in our model specification.

Furthermore, the fixed-domain semantics was introduced for DLs [GRS16]. By restricting in reasoning to an explicit domain, the semantics offers more intuitive models of some knowledge bases. Moreover, it gives complexity advantages in reasoning over expressive DLs. Explicit domains leads to more explicit models, and provokes a non-standard reasoning task, model enumeration. Combined with well-known existing tools for OWL, such as Protége, this approach provides a strong framework to build a model-oriented knowledge base.

Meanwhile, building a knowledge base is a process that is prone to errors. Naturally, it is a common task to find the cause of a problem that occurs in a knowledge base. One of the methods is finding minimal subsets of a given knowledge base that entail some consequences, which is called axiom pinpointing [PS09] or justification [Hor11]. While there are many terms coined for it, as the title suggests, we use the term justification from now on. There are several approaches used to do this task, e.g., automata-based [BP10a] and tableau-based [BP10b].

Thus, we looked at how the justification framework fits in fixed-domain semantics. We checked whether existing methods for finding justifications are adequate, or can be used with some tweaks to adapt it to the semantics. As stated in the previous work, it is possible to axiomatize a knowledge base such that all models are fixed models. Thus, we have a possible way to do the task in an uncomplicated way. While it might be possible to use them, we have to mind they are designed for standard semantics. They possibly are not optimized for axiomatization axioms that features a large nominal disjunction.
Consider an example which (deliberately) contains some faults.

**Example 1.1.** *(Animal Taxonomy Specification)* AT is a small animal taxonomy which also provides some more specified informations over them. AT taxonomy specifications are as follows: There are three types of animals: mammals, fishes and reptiles. Each type is disjoint to each other. There are two types of animals by their habitats: land and water habitat. All animals live either in land or water, but not both. All mammals live on land. All fishes live in water. All reptiles eat at least four types of fishes.

**Example 1.2.** *(Animal Taxonomy Data)* Whale is a mammal. Whale lives in water habitat. Crocodile is a reptile. Goldfish and catfish are fishes. Goldfish lives in water habitat. Crocodile eats goldfish.

Using a simple deduction, one can see the specification does not support the data. All mammals live on land and it is not possible that a land animal is a water animal at the same time. However, in the data we have whale, which is a water mammal. Furthermore, unlike in standard semantics, this knowledge specification is not consistent under certain fixed-domain. Consider the domain is fixed to all mentioned animals in the data, i.e. \( \Delta = \{ \text{whale, crocodile, goldfish, catfish} \} \). Now, the specification that any reptiles eat at least four types of fishes causes a problem too. The number of fishes that crocodile eats will not suffice. There are only four species in the domain, but we know whale and crocodile are not fishes.

We also looked to find another way to compute such task. The previous work for reasoning under fixed-domain semantics was implemented in Answer Set Programming (ASP) in a tool called Wolpertinger. ASP is based on the stable model semantics of logic programming \cite{GL88} and non-monotonic reasoning. ASP paradigm lies on modelling a problem into logic program, such that the answer sets of the program correspond to solutions of the original problem. The development, tools and feature for ASP keeps growing each year.

Thus, one of the promising solutions is introducing a new finding justifications method based on the translation in Wolpertinger. Furthermore, we tried to exploit ASP framework even more to do such task. Thus, we propose a translation from a given knowledge base to an ASP program such that one can extract the cause of the contradiction (if it occurs) from the ASP program models. There are quite many extensions of ASP itself. We used asprin *(ASP for preference handling)* \cite{BDRS15} which offers a greater control for model preferences to help such computation.

Currently, if we feed such knowledge base to Wolpertinger, the encoding ASP program is unsatisfiable, i.e., does not have any answer set. There is no more information about what is wrong with the original knowledge base. The good start for this is to provide the user with the justifications. In this work, we specifically look for inconsistency justification. We expect such algorithm will tell us that our specification that consists all mammals are land animals is a problem. The fix itself will depend on the designer of the taxonomy.
1.2 Problem Statement

The aforementioned motivation lead us to develop sophisticated and robust method to find justification under fixed-domain semantics, especially under Wolpertinger environment. Due to its nature, modelling is an error-prone process, either from human-error or logic-error. It is very hard to track all of axioms in the ontology to find the cause. Although there are many study regarding this problem, most of them are done under normal semantics. Thus, this thesis is written on several main ideas:

- To define the justification framework under fixed-domain semantics.
- To look and analyze for possible methods for finding justifications under fixed-domain semantics using existing tools.
- To develop a finding justifications method using ASP-based translation.
- To implement and evaluate such developed and applicable methods.

1.3 Thesis Outline

The thesis is structured as follows. First, we begin with preliminaries to describe the notions that we used for answer set programming and description logics $SROIQ$ in Chapter 2. It also contains the definition fixed-domain semantics which is a bit deeper than previous two well-known topics. In Chapter 3 we start going into the main topic, with existing notation of justification and bring them under fixed-domain semantics. We define a glass-box algorithm by encoding a knowledge base to an ASP program in Chapter 4. In Chapter 5 we make sure this encoding computes exactly what we want, i.e., justifications. We describe how the described methods are implemented and evaluated in Section 6. We conclude the work and look into possible future works in Chapter 7.
2 Preliminaries

The chapter provides some important concepts that are used as foundations of our work. We start by presenting description logic SROIQ in Section 2.1. We look a bit deeper into description logics by introducing the fixed-domain semantics in Section 2.2. A foundation for answer set programming can be found in Section 2.3.

2.1 Description Logics (SROIQ)

Description logics (DLs) are a family of knowledge representation language usually used in ontological modelling. They can be seen as decidable fragments of First Order Logic (FOL). Decidability is a key aspect which DLs provide while maintaining the expressiveness. The combination of them lead DLs into very popular frameworks in knowledge representation. Although some description logics have high complexity, e.g. ExpTime, some reasoners perform reasonably well in practice.

In this preliminary, we explain the highly expressive DL SROIQ [HKS06]. SROIQ is actually the underlying logic for OWL 2 if we leave the concrete domain \( D \). \( S \) stands for \( ALC \) (basic Description Logic) with transitive roles. \( R, O, I, Q \) are extensions for complex role inclusion, nominals, inverse properties and qualified cardinality restrictions, respectively. SROIQ extends SHOIN, the description logic underlying OWL-DL [SPG+07]. While SROIQ provides more expressiveness, it does not change decidability and practicability of SHOIN.

2.1.1 Syntax

We refer to [HKS06] and [KSH12] as the main resources of this formalization. We start by introducing the smallest part of vocabulary called \textit{signature}. Then, we proceed to define what expressions are allowed in SROIQ, and finally what knowledge base rules can be axiomatized.

**Definition 2.1.** (SROIQ Signature) Every (SROIQ) DL knowledge base is based on three finite and disjoint sets of signature: a set of individual names \( N_I \), a set of concept names \( N_C \) and a set of role names \( N_R \).

**Definition 2.2.** (Role Expressions) The set of SROIQ role expressions \( R \) is defined by the grammar

\[
R ::= U \mid N_R \mid N_R^-
\]  

(2.1)

where
2 Preliminaries

- \( U \) is the universal role which connects all pairs of individual,
- \( N_R = \{ r^- \mid r \in N \} \), \( r^- \) is called the inverse role of \( r \).

One of the strengths of \( SROIQ \) lies on the expressiveness in describing roles. For example, \( SROIQ \) allows for disjoint roles axiom, where most DLs only allow to express disjointness on concepts. Furthermore, \( SROIQ \) provides complex role inclusion axiom of the form \( r \circ s \subseteq r \) and \( s \circ r \subseteq s \). However, arbitrary property chain axioms lead to undecidability. Thus, we need to introduce the notions of regularity and simplicity.

Definition 2.3. (Regular Role Inclusion Axioms) Let \( \prec \) be a regular order on roles. A role inclusion axiom (RIA) is an expression of the form \( s_1 \circ \ldots \circ s_n \subseteq r \), where any of \( s_i \) and \( r \) is a role name, but not \( U \). A role hierarchy \( R_h \) is a finite set of RIAs. A RIA \( S_1 \circ \ldots \circ s_n \subseteq r \) is \( \prec \)-regular if \( r \) is a role name, and of the form

- \( r \circ r \subseteq r \), or
- \( r^- \subseteq r \), or
- \( s_1 \circ \ldots \circ s_n \subseteq r \) and \( s_i \prec r \), for all \( 1 \leq i \leq n \), or
- \( r \circ s_1 \circ \ldots \circ s_n \subseteq r \) and \( s_i \prec r \), for all \( 1 \leq i \leq n \), or
- \( s_1 \circ \ldots \circ s_n \circ r \subseteq r \) and \( s_i \prec r \), for all \( 1 \leq i \leq n \).

A role hierarchy \( R_h \) is regular if there exists a regular order \( \leq \) such that each RIA in \( R_h \) is \( \leq \)-regular.

Definition 2.4. (Role Assertions) A role assertion is one of the following forms: \( Ref(r) \) (Reflexive), \( Irr(r) \) (Irreflexive), \( Sym(r) \) (Symmetric), \( Asy(r) \) (Asymmetric), \( Tra(r) \) (Transitive), and \( Dis(r, s) \) (Disjoint) for \( r, s \neq U \) are roles.

Definition 2.5. (Simple Role) Let \( R_h \) be a role hierarchy and \( R_a \) be a set of role assertions. The definition of non-simple role expressions is given by the following rules:

- if \( R_h \) contains an axiom \( s \circ t \subseteq r \) then \( r \) is non-simple.
- if \( r \) is non-simple, then its inverse \( r^- \) is also non-simple.
- if \( r \) is non-simple and \( R_h \) contains any of the axiom \( r \subseteq s \), \( s \equiv r \) or \( r \equiv s \), then \( s \) is also non-simple.

All other roles are called simple. \( R_a \) is simple if for any \( Irr(r) \), \( Asy(r) \), or \( Dis(r, s) \), \( r \) and \( s \) are simple in \( R_h \).

Definition 2.6. (Role Box) A \( SROIQ \)-role box (RBox) is a set of \( R = R_h \cup R_a \), where \( R_h \) is a regular role hierarchy and \( R_a \) is a finite, set of role assertions.

After having a (restricted) definition of the roles, \( SROIQ \)-concepts can be defined. We will see that there are several restrictions over the roles that can be used for some concept constructors.
2.1 Description Logics (SROIQ)

**Definition 2.7.** (Concept Expression) The set of SROIQ concept expression $C$ is defined by the grammar

$$C ::= N_C | (C \sqcap C) | (C \sqcup C) | \neg C | \top | \bot | \exists r.C | \forall r.C | \geq n.s.C | \leq n.s.C | \exists Self | \{a\}$$

where

- $\top$ is the top, special concept with every individual as an instance,
- $\bot$ is the bottom, dual concept of $\top$ with no individual,
- $r$ is a role,
- $s$ is a simple role.

* parentheses may be omitted if there is no ambiguity.

Now, we have enough building blocks to define axioms of SROIQ. As usual, we have the terminological box and assertional box. We define what a terminological box is first, since it represents hierarchy and relation between concepts in the knowledge base.

**Definition 2.8.** (General Concept Inclusion) A general concept inclusion (GCI) is an expression of the form $C \sqsubseteq D$ for two SROIQ-concepts $C$ and $D$. $C \equiv D$ stands for $C \sqsubseteq D$ and $D \sqsubseteq C$.

**Definition 2.9.** (Terminological Box) A SROIQ Terminological Box (TBox) $T$ is a finite set of GCIs.

We fill the knowledge base with assertions using an assertional box (ABox). It handles known concept assertions, role assertions and (in)equality assertions. An assertional box represents known data.

**Definition 2.10.** (Individual Assertion) An individual assertion is one of the following forms: $C(a)$, $R(a,b)$, $a \approx b$, or $a \not\approx b$, for $a, b \in NI$, a (possibly inverse) role $R$, and a SROIQ-concept $C$.

**Definition 2.11.** (Assertional Box) A SROIQ Assertional Box (ABox) $A$ is a finite set of individual assertions.

**Definition 2.12.** (Knowledge Base) A SROIQ Knowledge Base (KB) $K = (T, A, R)$ consists of a TBox $T$, an ABox $A$, and RBox $R$.

2.1.2 SROIQ Semantics

Intuitively, the semantics of DLs defines individual membership over the concepts and relation between individuals for roles. Formally, we define

**Definition 2.13.** (Interpretation) A (SROIQ) interpretation is a tuple $I = (\Delta^I, \cdot^I)$ where $\Delta^I$ is a non-empty set called domain of $I$, and $\cdot^I$ is a function that maps...
• every individual \( a \) to an element \( a^\mathcal{I} \in \Delta^\mathcal{I} \),
• every concept to a subset of \( \Delta^\mathcal{I} \), and
• every role name to a subset of \( \Delta^\mathcal{I} \times \Delta^\mathcal{I} \).

**Definition 2.14.** (Semantics) Given an interpretation \( \mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I}) \), concepts \( C, D \), role \( R \), and non-negative integer \( n \), the extension of complex concept interpretation is defined inductively in Table 2.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>inverse role</td>
<td>( s^- )</td>
<td>( { (x, y) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid (y, x) \in s^\mathcal{I} } )</td>
</tr>
<tr>
<td>universal role</td>
<td>( u )</td>
<td>( \Delta^\mathcal{I} \times \Delta^\mathcal{I} )</td>
</tr>
<tr>
<td>top</td>
<td>( \top )</td>
<td>( \Delta^\mathcal{I} )</td>
</tr>
<tr>
<td>bottom</td>
<td>( \bot )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>negation</td>
<td>( \neg C )</td>
<td>( \Delta^\mathcal{I} \setminus C^\mathcal{I} )</td>
</tr>
<tr>
<td>conjunction</td>
<td>( C \cap D )</td>
<td>( C^\mathcal{I} \cap D^\mathcal{I} )</td>
</tr>
<tr>
<td>disjunction</td>
<td>( C \cup D )</td>
<td>( C^\mathcal{I} \cup D^\mathcal{I} )</td>
</tr>
<tr>
<td>nominals</td>
<td>( { a_1, \ldots, a_n } )</td>
<td>( { a_1^\mathcal{I}, \ldots, a_n^\mathcal{I} } )</td>
</tr>
<tr>
<td>univ. restriction</td>
<td>( \forall r.C )</td>
<td>( { x \mid \forall y.(x, y) \in r^\mathcal{I} \rightarrow y \in C^\mathcal{I} } )</td>
</tr>
<tr>
<td>exist. restriction</td>
<td>( \exists r.C )</td>
<td>( { x \mid \exists y.(x, y) \in r^\mathcal{I} \land y \in C^\mathcal{I} } )</td>
</tr>
<tr>
<td>Self concept</td>
<td>( \exists r.\text{Self} )</td>
<td>( { x \mid (x, x) \in r^\mathcal{I} } )</td>
</tr>
<tr>
<td>qualified number</td>
<td>( \leq n r.C )</td>
<td>( { x \mid # { y \in C^\mathcal{I} \mid (x, y) \in r^\mathcal{I} } \leq n } )</td>
</tr>
<tr>
<td>restriction</td>
<td>( \geq n r.C )</td>
<td>( { x \mid # { y \in C^\mathcal{I} \mid (x, y) \in r^\mathcal{I} } \geq n } )</td>
</tr>
</tbody>
</table>

Table 2.1: Syntax and semantics of role and concept constructors in SROIQ, where \( a_1, \ldots, a_n \) denote individual names, \( s \) a role name, \( r \) a role expression and \( C \) and \( D \) concept expressions.

We write \( \mathcal{I} \models \alpha \) for \( \mathcal{I} \) satisfies an axiom \( \alpha \). \( \mathcal{I} \) satisfies GCI \( C \sqsubseteq D \) iff \( C^\mathcal{I} \subseteq D^\mathcal{I} \). \( \mathcal{I} \) is a model of a TBox \( \mathcal{T} \) if \( \mathcal{I} \) satisfies each GCI in \( \mathcal{T} \). \( x \in \Delta^\mathcal{I} \) is called an instance of a concept \( C \) if \( x \in C^\mathcal{I} \). \( \mathcal{I} \) satisfies an individual assertion of the form:

- \( C(a) \) if \( a^\mathcal{I} \in C^\mathcal{I} \),
- \( r(a, b) \) if \( (a^\mathcal{I}, b^\mathcal{I}) \in r^\mathcal{I} \),
- \( a \approx b \) if \( a^\mathcal{I} = b^\mathcal{I} \), and
- \( a \not\approx b \) if \( a^\mathcal{I} \neq b^\mathcal{I} \).

\( \mathcal{I} \) satisfies (is a model of) ABox \( \mathcal{A} \) if \( \mathcal{I} \) satisfies each \( \alpha \in \mathcal{A} \). We define for an interpretation \( \mathcal{I} \), \( \text{Diag}^\mathcal{I} = \{ (x, x) \mid x \in \Delta^\mathcal{I} \} \). \( \mathcal{I} \) satisfies a role assertion of the form:

- \( \text{Sym}(r) \) if \( (x, y) \in r^\mathcal{I} \) implies \( (y, x) \in r^\mathcal{I} \);
- \( \text{Asy}(r) \) if \( (x, y) \in r^\mathcal{I} \) implies \( (y, x) \notin r^\mathcal{I} \).
2.1 Description Logics (SROIQ)

- \textit{Tra}(r) if \((x, y) \in r^I \text{ and } (y, z) \in r^I\) implies \((x, z) \in r^I\);
- \textit{Ref}(r) if \(\text{Diag}^I \subseteq r^I\);
- \textit{Irr}(r) if \(r^I \cap \text{Diag}^I = \emptyset\);
- \textit{Dis}(r, s) if \(r^I \cap s^I = \emptyset\).

An interpretation \(I\) satisfies \(KB = (T, A, R)\) if \(I\) satisfies any axiom \(\alpha \in K\). Knowledge base \(K\) is satisfiable (consistent) if there is such an interpretation, and unsatisfiable (inconsistent) conversely.

It can be shown, that it is possible to rewrite any of transitivity, a(symmetry), and (ir)reflexivity axioms using disjointness and role inclusions axiom (syntactic sugar).

2.1.3 Reasoning Tasks

There are several standard tasks for both terminological and assertional reasoning. For terminological reasoning, Our only concern is the TBox. Meanwhile in assertional reasoning, we take the whole ontology as the input.

\begin{definition}
\textit{(Terminological Reasoning)} Let \(T\) be a TBox, \(C, D\) be concepts.
\begin{itemize}
\item Satisfiability: \(C\) is satisfiable w.r.t \(T\) iff \(C^I \neq \emptyset\) for some model \(I\) of \(T\).
\item Subsumption: \(C\) is subsumed by \(D\) w.r.t. \(T\) \((C \sqsubseteq_T D)\) iff \((C^I \subseteq D^I)\) for all models \(I\) of the TBox \(T\).
\item Equivalence: \(C\) is equivalent to \(D\) w.r.t. \(T\) \((C \equiv_T D)\) iff \((C^I = D^I)\) for all models \(I\) of the TBox \(T\).
\end{itemize}
\end{definition}

\begin{definition}
\textit{(Assertional Reasoning)} Let \(K = (T, A, R)\) be a SROIQ knowledge base.
\begin{itemize}
\item Consistency: \(K\) is consistent iff there exists a model of \(K\).
\item Instance: \(a\) is an instance of \(C\) iff \(a^I \in C^I\) for all models \(I\) of \(K\).
\end{itemize}
\end{definition}

In the standard semantics, there is a clear boundary between terminological and assertional reasoning tasks. For example, the subsumption task does not consider any individual assertion in ABox since it will not change anything. However, as we consider different semantics in this work, we will see those assertions may affect a subsumption. We also have non-standard reasoning tasks, e.g. finding justifications. Moreover, we deal with justifications as the main topic of this work, therefore we will introduce it in the upcoming section.

While the exact complexity of SROIQ is not pointed in the original paper, it has been shown in [Kaz15] that it is harder than SHOIN, the language corresponds to OWL-DL. The exponential growth is due to complex role inclusion axiom. The satisfiability problem of SROIQ ontologies is \textbf{N2ExpTime}-complete, and consequently so are other all the standard reasoning problems.
2 Preliminaries

2.1.4 Web Ontology Language (OWL)

The Web Ontology Language (OWL) is a Semantic Web language that provides a standard for representing and exchanging knowledge over the web. OWL is used mainly to describe hierarchy of concepts and the relation between them and their instances. Furthermore, it is used to provide clarity over a knowledge, for example to remove ambiguity between two instances with same name. It is tailored not only to represent, but also to process and deduce a new fact from a defined knowledge base.

In fact, OWL is strongly related to Description Logics. Reasoning over OWL is reasoning over its corresponding DL knowledge base. Each OWL syntax corresponds to a family of Description Logics. OWL 2 is the current standard which corresponds to the description logic SROIQ. Although OWL 2 came with more features, e.g. datatypes, we stay with SROIQ in this work. Several OWL reasoners have been developed, for example Pellet [SPG07], Konclude [SLG14], and HermiT [MSH09]. One of the common tools used for working with OWL is Protégé [Mus15], a UI-based ontology editor built with Java. It also provides HermiT as a built-in reasoner to extract further knowledge from developed ontologies.

For example, we show a syntax in OWL and the corresponding axiom in DL. We want to restrict that the range of the property (role in DL) hasFather should be a Man.

\[
\text{hasFather rdfs:range :Man} \iff \top \sqsubseteq \forall \text{hasFather.Man}
\]

This example also shows OWL has more friendly syntax, but still can be represented by SROIQ which defined in Section 2.1. The full details of OWL 2 specification is described in [W3C09].

2.2 Fixed Domain Semantics [GRS16]

Fixed domain semantics can be seen as further restriction for finite model reasoning. While finite model reasoning consider any arbitrary finite domain, we restrict furthermore with a known domain. This restriction gives us not only a computation complexity advantage, but arguably more intuitive models of a knowledge base in some cases.

Definition 2.17 (Fixed-Domain Semantics). Let \( \Delta \) be a non-empty finite set called fixed domain. An interpretation \( \mathcal{I} = (\Delta^\mathcal{I}, \mathcal{T}) \) is said to be \( \Delta \)-fixed, if

- \( \Delta^\mathcal{I} = \Delta \), and
- \( a^\mathcal{I} = a \) for all \( a \in \Delta \).

For a DL knowledge base \( \mathcal{K} \), we call an interpretation \( \mathcal{I} \) a \( \Delta \)-model of \( \mathcal{K} \) (\( \mathcal{I} \models_\Delta \mathcal{K} \)), if \( \mathcal{I} \) is a \( \Delta \)-fixed interpretation and \( \mathcal{I} \models \mathcal{K} \). A knowledge base \( \mathcal{K} \) is \( \Delta \)-satisfiable if there exists a \( \Delta \)-model. A knowledge base \( \mathcal{K} \) \( \Delta \)-entails an axiom \( \alpha \) (\( \mathcal{K} \models_\Delta \alpha \)) if \( \mathcal{I} \models \alpha \) for every \( \mathcal{I} \models \Delta \mathcal{K} \).
We know that fixed domain reasoning gives stronger restrictions than standard and finite model reasoning. Thus, it is easy to see that for any knowledge base $KB$, a fixed-domain $\Delta$, and an axiom $\alpha$ that $\mathcal{K} \models \alpha$ implies $\mathcal{K} \modelsfin \alpha$, and $\mathcal{K} \modelsfin \alpha$ implies $\mathcal{K} \models\Delta \alpha$.

**Example 2.18.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A}, \mathcal{R})$, where $\mathcal{T} = \{A \subseteq \exists r.A, A \subseteq 1 - r^- A\}$, $\mathcal{A} = \{A(b), B(a), B(b)\}$, and $\mathcal{R} = \emptyset$. We define a fixed domain $\Delta = \{a, b\}$. We may see the example represents a chain of role $r$. Consider an axiom $\alpha$, where

- $\alpha = \top \subseteq \exists r.\top$. Then, $\alpha$ holds in all models (under standard semantics) of $\mathcal{K}$. Thus, $\mathcal{K} \models \alpha$, $\mathcal{K} \modelsfin \alpha$, and $\mathcal{K} \models\Delta \alpha$.

- $\alpha = A(a)$. Consider we have finite models only. Then, the “last individual” in the $r$-chain should be connected to individual $a$. Thus, $\alpha$ holds in all finite models. Thus, $\mathcal{K} \not\models \alpha$, $\mathcal{K} \modelsfin \alpha$, and $\mathcal{K} \models\Delta \alpha$.

- $\alpha = A \subseteq B$. In a finite model, individual $b$ may be connected to some other individuals until eventually connected to $a$. These individuals does not have to be in the concept $A$. Thus, $\mathcal{K} \not\models \alpha$, $\mathcal{K} \not\modelsfin \alpha$, and $\mathcal{K} \models\Delta \alpha$.

Now, we may ask how to draw a relation between fixed domain semantics with the standard one. Indeed, it is possible to restrict a knowledge base with standard semantics so that we eliminate all non-$\Delta$-fixed models. Intuitively, we restrict the domain such that there is no other individual appears in a model and ensure the mapping of any individual is to itself.

**Theorem 2.19.** Let $\Delta = \{a_1, ..., a_n\}$ be a (fixed)-domain and $\mathcal{K}$ be a DL knowledge base. We define

$$
\mathcal{FD}_\Delta := \{\top \subseteq \{a_1, ..., a_n\}\} \cup \{a_i \neq a_j \mid i < j \leq n\}.
$$

$\mathcal{K}$ is $\Delta$-satisfiable iff $\mathcal{K} \cup \mathcal{FD}_\Delta$ is satisfiable (under standard semantics). For any axiom $\alpha$, $\mathcal{K} \models\Delta \alpha$ iff $\mathcal{K} \cup \mathcal{FD}_\Delta \models \alpha$.

It has been shown that fixed-domain reasoning has a complexity advantage over standard reasoning. The main idea of $NP$-membership proof is showing we can guess the interpretation extension over the domain, and then checking consistency of every axiom in the knowledge base using this interpretation. Fixed-domain satisfiability checking in any language between $DL_{\min}$ and $SROIQ$ is $NP$-complete.

This $NP$-completeness of fixed-domain reasoning gave not only a complexity advantage, but also the algorithm development perspective. The nature of $NP$ problem algorithm, which generates a solution and then verifies it, coincides with the “guess and check” ASP paradigm.
2 Preliminaries

2.3 Answer Set Programming

Answer set programming (ASP) is a form of declarative programming oriented towards difficult, primarily NP-hard, search problem \cite{Lif08}. ASP can be seen as logic programming under the stable model semantics \cite{GL88}. It is mainly used to tackle combinatoric problems by modelling them into logic programs and use an answer set solver to compute stable models. It is more focused on models rather than proofs. Thus, an ASP solver is about the technique to compute models.

Despite having rules that look like Prolog rules, ASP is purely declarative. The order of program rules does not matter in ASP, so do the subgoals in a rule. Furthermore, an ASP program always terminates. There is no function in (standard) ASP solvers. It is in fact quite close to SAT where answer sets are particular classical models of the program. However, ASP provides more expressive and high-level features, such as transitive closures, negation as failure, and variables. The following preliminary for ASP is based on some previous works \cite{EIK09,GL88}.

2.3.1 Syntax

An ASP program is built upon a first-order language $\mathcal{L}$, hence shares many terminologies with classical logic. For the smallest part, we have

- variables which denoted by upper-case letters, usually $X, Y, Z$, etc.
- constants which denoted by lower-case letters, usually $a, b, c$, etc.
- predicates which denoted by lower-case letters, usually $p, q, r$, etc.
- functions which denoted by lower-case letters, usually $f, g, h$, etc.

**Definition 2.20.** (Terms, Atoms and Literals) A term is a variable, a constant, or an expression of the form $f(t_1, ..., t_n)$ where $f$ is a function and each $t_1, ..., t_n$ is a term. An atom is an expression of the form $p(t_1, ..., t_n)$, where $p$ is an $n$-ary predicate, and each $t_1, ..., t_n$ is a term. A strongly negated atom is an expression of the form $\neg A$, where $A$ is an atom. The brackets are usually omitted when $n = 0$. A classical atom is an atom or a strongly negated atom. A built-in atom has form $t < u$ for terms $t$ and $u$ with $\prec \in \{", <", ", \leq", ", =", ", >", ", \geq"\}$. Built-in atoms $a$, as well as the expressions $a$ and $\text{not } a$ for a classical atom $a$ are naf-literals.

**Definition 2.21.** (Aggregate Literals) An aggregate element has form $t_1, ..., t_m : l_1, ..., l_n$ where $t_1, ..., t_m$ are terms and $l_1, ..., l_n$ are literals with $m \geq 0$ and $n \geq 0$. An aggregate atom has form $\#aggr\{e_1; ..., e_n\}$ where $e_1, ..., e_n$ are aggregate elements for $n \geq 0$, $\#aggr \in \{"\text{count}"", "\text{sum}"", "\text{max}"", "\text{min}"\}$ is an aggregate function name, $\prec \in \{", <", ", \leq", ", =", ", >", ", \geq"\}$ is an aggregate relation and $u$ is a term. The expressions $a$ and $\text{not } a$ for an aggregate atom $a$ are aggregate literals.

There are several extensions of ASP. Weakening the rule, hence supporting more expressive statements, may cause a higher complexity. In this work, we use Extended Disjunctive Logic Programs (EDLP).
Definition 2.22. *Extended Disjunctive Logic Program* An extended disjunctive logic program (EDLP) is a set of rules of the form:

\[ a_1 \lor \cdots \lor a_n \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m, \]

with \( k, n, \geq 0 \), where all \( a_i, b_i \) are classical atoms, or aggregate atoms. \textit{not} denotes default negation, or known as negation as failure (NAF) \([Cla78]\). We denote each part of the rule with \( \text{head}(r) \) for \( \{a_1, \ldots, a_n\} \), \( \text{pos}(r) \) for \( \{b_1, \ldots, b_n\} \), and \( \text{neg}(r) \) for \( \{b_{k+1}, \ldots, b_m\} \). Finally, to collect all of atoms that appear in a rule, we define \( \text{lit}(r) = \text{head}(r) \cup \text{pos}(r) \cup \text{neg}(r) \).

An Extended Logic Program (ELP) does not have disjunction in the head, i.e., \( n = 1 \). A Normal Logic Program (NLP) is an ELP that does not have any strongly negated atoms. However, strong negation is a syntactic sugar feature of NLP, thus an ELP can be reduced to an NLP. Furthermore, we introduce an extension called \textit{choice rules}.

Definition 2.23. *Choice Rules* A choice rule is of the form

\[ \{a_1, \ldots, a_m\} \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m, \]

with \( k, n, \geq 0 \), where all \( a_i, b_i \) are classical atoms, or aggregate atoms.

Choice rules are syntactic sugar of EDLP and can be translated into \( 2m + 1 \) normal rules. The meaning of the rule is any subset of \( \{a_1, \ldots, a_m\} \) can be included in the stable model if the body is satisfied. This rule can be used for generating or guessing solution in a compact way.

2.3.2 Semantics

While the syntax allows the use of variables, we define the semantics of ASP programs without any variable. We define a process of removing those variables called \textit{grounding}, which make use of \textit{Herbrand Interpretation}.

Definition 2.24. *Herbrand Universe, Base, Interpretation* Given a logic program \( \Pi \), the Herbrand universe of \( \Pi \), \( \text{HU}(\Pi) \), is the set of all terms which can be formed from constants and functions symbols in \( \Pi \) (resp. the vocabulary of \( \mathcal{L} \), if explicitly known). The Herbrand base of \( \Pi \), \( \text{HB}(\Pi) \), is the set of all ground atoms which can be formed from predicates occurring in \( \Pi \) and the terms in \( \text{HU}(\Pi) \). A (Herbrand) interpretation is an interpretation \( I \) over \( \text{HU}(\Pi) \), that is, \( I \) as subset of \( \text{HB}(\Pi) \).

Note that not all ASP system have function symbols features. The ASP solver used in this work, \texttt{Clingo}, has function symbols feature. However, any unbounded function might produce an infinite grounding and not halt. An example of such case is the rule \( p(f(f(X))) :- p(f(X)). \)

Definition 2.25. *Grounding* A ground instance of a clause \( C \) is any clause \( C' \) obtained from \( C \) by applying a substitution \( \theta : \text{Var}(C) \rightarrow \text{HU}(\Pi) \) to the variables in \( C \), denoted as \( \text{Var}(C) \). For any clause \( C \), we denote \( \text{grnd}(C) \) the set of all possible ground instances of \( C \), and for any program \( \Pi \) we let \( \text{grnd}(\Pi) = \bigcup_{C \in \Pi} \text{grnd}(C) \) (the grounding of \( \Pi \)).
2 Preliminaries

Definition 2.26. (Reduct) The Gelfond-Lifschitz reduct (reduct) of an EDLP program $\Pi$ w.r.t. an interpretation $M$, denoted $\Pi^M$, is a program obtained by

1. removing rules with not $a$ in the body for each $a \in M$;
2. removing literals not $a$ from all other rules.

Definition 2.27. (Model) Let $M$ be an interpretation. $M$ is a model of:

- a ground clause $C : a_1 \lor \ldots \lor a_n \leftarrow b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m$, denoted $M \models C$, if $\{b_1, \ldots, b_k\} \not\subseteq M$ or $\{a_1, \ldots, a_n, b_{k+1}, \ldots, b_m\} \cap M \neq \emptyset$.
- a clause $C$, denoted $M \models C$, if $M \models C'$ for every $C' \in \text{grnd}(C)$.
- a program $\Pi$, denoted $M \models \Pi$, if $M \models C$ for every clause $C \in \Pi$.

Definition 2.28. (Stable Model) An interpretation $M$ of $\Pi$ is a stable model of $\Pi$, if $M$ is a minimal model of $\Pi^M$ (w.r.t. $\subseteq$).

EDLP is $\Sigma_p^2$-hard which can be shown by reduction of validity from $QBF_{2,3}$ [EG95]. Meanwhile, NLP is $NP$-complete which means EDLP is not just a syntactic sugar of NLP.

2.3.3 Examples

Consider a simple example:

$$ a \lor b \lor c \leftarrow $$

There are six models of the rule (consequently the program): $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, $\{a, b, c\}$. Due to the minimality, the answer sets of the program are $\{a\}$, $\{b\}$, $\{c\}$. Empty set ($\emptyset$) is not a model because it does not satisfy the rule.

It is possible that two answer sets share the same atoms in a disjunction. We consider another simple example:

$$ a \lor b \leftarrow 
\quad b \lor c \leftarrow 
\quad a \lor c \leftarrow $$

The answer sets for this example are $\{a, b\}$, $\{b, c\}$, and $\{a, c\}$. Obviously, they are minimal, thus not contained in each other. However, we could see that both first and second answer sets contain $b$. This example shows that disjunction does not mean being exclusive.
2.3.4 Guess and Check Modeling

In ASP paradigm, one should reduce a problem instance to computing stable models of an LP. We encode a problem into a logic program, such that the solutions of the problem are represented by models of the logic program. Then, we compute the model of the logic program and extract the solution from it. Multiple solutions can be acquired from non-determinism of ASP.

Furthermore, we separate the problem into specification logic and instance data. For a concrete example, consider solving a sudoku problem. The specification logic should be defined based on the sudoku rules, e.g. there is no duplicate number in the same line. Until this point, the answer sets are all the possible combinations of filling an empty sudoku board. This enormous answer sets will be pruned by the data provided based on the instance.

An important methodology in writing an ASP program is “Guess and Check”. The specification part is divided into guessing part and checking part. The guessing part represent solution candidate for the problem, whereas the checking part use further rules and constraints to specify the solution requirements. This methodology can be extended by optimization part when the problem has weak constraints, such that some answer sets are preferred over others.

Consider 3-coloring problem as an example. Given a graph $G = (V, E)$, we need to color every vertex with three colors (e.g. red, green, blue) such that there is no pair of connected nodes that has the same color. Let $n(X)$ represents $X$ is a node, $e(X,Y)$ represents an edge between $X$ and $Y$, and $r(X), g(X), b(X)$ represents $X$ has color red, green, or blue respectively.

\[
\begin{align*}
    r(X) \lor g(X) \lor b(X) & \leftarrow n(X) \\
    & \leftarrow r(X), r(Y), e(X,Y) \\
    & \leftarrow g(X), g(Y), e(X,Y) \\
    & \leftarrow b(X), b(Y), e(X,Y)
\end{align*}
\]

It is easy to see the guessing part is the first line, while the rest is the checking part. The checking part simply removes models where two nodes connected by an edge have the same color. This example also shows the use of disjunction for more intuitive guessing.
3 Justifications under Fixed-Domain Semantics

In this chapter, we describe an existing foundation of justification framework for DL knowledge bases. Then, we define and analyze the justification framework under fixed-domain semantics.

3.1 Justification

Building a DL knowledge base, or syntactically OWL, is prone to error. Looking where the error is, is not an easy task. A set of axioms that entails a specific axiom entailment is called a justification. It is possible that there are more than one justification for one entailment in an ontology. We borrow the formalization of justification and simple repair from [HPS09a].

Definition 3.1 (Justification). Let $O$ be an ontology such that $O \models \eta$. $J$ is a justification for $\eta$ in $O$ if $J \subseteq O$ and $J \models \eta$, and for all $J' \subset J$, $J' \not\models \eta$.

The last part of Definition 3.1 says not all of the subsets where the axiom still holds are interesting, but only the minimal ones. Assuming monotonicity, for any justification $J$, all of the supersets $J'' \supset J$, $J'' \models \eta$ holds. Furthermore, removing one of the axioms in each justification will eliminate the entailment of $\eta$ from the ontology.

Example 3.2 (Justification Example). Let $O = \{A \sqsubseteq B, A \sqsubseteq \neg B, A \sqsubseteq (C \sqcap \neg C), C(a)\}$. $A$ is unsatisfiable in $O$, i.e. $O \models A \sqsubseteq \bot$. There are two justifications for $O \models A \sqsubseteq \bot$, $J_1 = \{A \sqsubseteq B, A \sqsubseteq \neg B\}$ and $J_2 = \{A \sqsubseteq (C \sqcap \neg C)\}$.

The first justification is the occurrence of two concept subsumption and disjointness together. Second justification consists only of one axiom which cause any member of $A$ should be in $C$ and $\neg C$ at the same time. Adding ABox assertion $A(a)$ to $O$ causes an inconsistency, i.e. $O \models \bot(a)$. The justification of this inconsistency is any of the previous justifications appended with $A(a)$.

Definition 3.3 (Simple Repair). $S$ is a simple repair for $O \models \eta$, if $S \subseteq O$ and, for every justification $J \models \eta$, $S \cap J \neq \emptyset$, and for all $S' \subset S$, $(O \setminus S') \models \eta$.

While the original definition uses the term ontology, from now on we will use term knowledge base. Furthermore, if the context is clear we may describe a knowledge base $K$ as a single set of axioms, where $K = T \cup A \cup R$. Finding justifications is not an easy task, as there are exponentially many subsets to be checked. Generally, there are two types of justification algorithms, black-box and glass-box [KPHS07].
• A black-box algorithm uses the reasoner as an independent resource. The reasoner can be seen as a "black-box" to answer queries, hence the term. The process of looking for the justifications is handled outside of the reasoner and does not change any part of it. Usually, black-box algorithms feed an ontology subset as a guess to the reasoner and ask for consistency. The selection of which subset to be guessed determines the performance of the algorithm. Usually they are more robust and easy to implement.

• A glass-box algorithm has a high dependency with how the reasoner works. It modifies the existing reasoner algorithm to find justifications. This approach hopes for a better efficiency as it really looks into how the reasoner does the specific task and optimizes it for finding justifications.

An example of black-box algorithm is an approach using Reiter’s Hitting Set Tree (HST). Originally defined for theory of diagnosis, Reiter’s HST provides more sophisticated subset guessing for searching justifications. The details of the algorithm can be found in [KPHS07].

While justification framework is defined for any arbitrary axiom, we focused on finding justifications for inconsistency in this work. We could axiomatize inconsistency using axiom $\top \sqsubseteq \bot$. In this way, we could see a justification for $\top \sqsubseteq \bot$ is the minimal inconsistent subset of an ontology.

### 3.1.1 Example

We look back at our examples in Chapter 1 about the animal taxonomy. First, we translate specifications in Example 1.1 into a TBox $\mathcal{T}_{AT}$.

**Example 3.4. (AT Specification TBox)**

$$
\mathcal{T}_{AT} = \{\text{Mammal} \sqsubseteq \text{Animal}, \text{Fish} \sqsubseteq \text{Animal}, \text{Reptile} \sqsubseteq \text{Animal}\} \cup \\
\{\text{Mammal} \sqsubseteq \neg \text{Fish}, \text{Fish} \sqsubseteq \neg \text{Reptile}, \text{Reptile} \sqsubseteq \neg \text{Mammal}\} \cup \\
\{\text{LandAnimal} \sqsubseteq \text{Animal}, \text{WaterAnimal} \sqsubseteq \text{Animal}\} \cup \\
\{\text{Animal} \sqsubseteq \text{LandAnimal} \sqcap \text{WaterAnimal}\} \cup \\
\{\text{LandAnimal} \sqsubseteq \neg \text{WaterAnimal}\} \cup \\
\{\text{Mammal} \sqsubseteq \text{LandAnimal}, \text{Fish} \sqsubseteq \text{WaterAnimal}\} \cup \\
\{\text{Reptile} \sqsubseteq \geq 4 \text{ eat.Fish}\}.
$$

Then, we translate the data specified in Example 1.2 to an ABox $\mathcal{A}_{AT}$. We turn most of them into concept assertions, except the last one that is translated to a role assertion. Of course, we keep the signature over this ABox $\mathcal{A}$ to keep it in line with the signature introduced in the TBox $\mathcal{T}_{AT}$. 

18
3.1 Justification

Example 3.5. \( (AT \text{ Data ABox}) \)

\[
\mathcal{A}_{AT} = \{ \text{Mammal}(\text{whale}), \text{WaterAnimal}(\text{whale}), \text{Reptile}(\text{crocodile}) \} \cup \\
\{ \text{Fish}(\text{goldfish}), \text{Fish}(\text{catfish}), \text{WaterAnimal}(\text{goldfish}) \} \cup \\
\{ \text{eat}(\text{crocodile}, \text{goldfish}) \}. \quad (3.1)
\]

We do not have any specification to be translated into a role axiom, thus \( \mathcal{R}_\text{Box} \) is empty. We collect all of the information into a knowledge base \( \mathcal{K}_{\text{AT}} \).

Example 3.6. \( (AT \text{ Knowledge Base}) \)

\[
\mathcal{K}_{\text{AT}} = (\mathcal{T}_{\text{AT}}, \mathcal{A}_{\text{AT}}, \emptyset). \quad (3.4)
\]

Note that this translation is not the only translation and arguably a good one. For example, we introduce concepts \( \text{LandAnimal} \) and \( \text{WaterAnimal} \) rather than introducing the concept \( \text{Habitat} \). Choosing what are the concepts and what are the individuals is a modelling problem and depends on the domain we want to talk about. Nevertheless, we stay with the example above to show the problem of the ontology later.

Furthermore, \( \mathcal{K}_{\text{AT}} \) is inconsistent, i.e. has no model. We know that \( \text{Mammal}(\text{whale}) \), hence \( \text{LandAnimal}(\text{whale}) \) because of \( \text{Mammal} \sqsubseteq \text{LandAnimal} \). However, we know that \( \text{LandAnimal} \) and \( \text{WaterAnimal} \) are disjoint, or formally we can conclude that \( \neg \text{WaterAnimal}(\text{whale}) \) holds. Concurrently, we know \( \text{WaterAnimal}(\text{whale}) \) holds from the data, hence a contradiction. We formalize this explanation using the justification framework.

Example 3.7. \( (AT \text{ Knowledge Base Inconsistency Justification}) \) Let \( \eta = \top \sqsubseteq \bot \) (represents inconsistency). We obtain

\[
\mathcal{J}_1 = \{ \text{Mammal}(\text{whale}), \text{WaterAnimal}(\text{whale}) \} \cup \quad (3.5)
\]

\[
\{ \text{Mammal} \sqsubseteq \text{LandAnimal}, \text{LandAnimal} \sqsubseteq \neg \text{WaterAnimal} \}. \quad (3.6)
\]

\( \mathcal{J}_1 \) is a justification for \( \eta \) in \( \mathcal{K}_{\text{AT}} \).

3.1.2 Working with Justification

We would like to discuss what we can do in practice with justifications. Obviously, justifications can be used to find out where are the problems of an inconsistent ontology. While it sounds great, there are some problems in the further concrete application. In fact, it is sometimes not feasible to find all justifications or the justification is still too complicated to be understood by humans on abstract level. For the latter, it is naturally easier (not always) to understand justifications with small size.

While finding all justifications is a hard task, it is still very useful to get several of them. In this way, we can do an incremental repair to the ontology. It is also possible that a justification can give a hint what other justifications might be for human. In a sense it may be worth to ask for some justifications instead of all of them. It is also
possible to set a timeout when asking for justifications. Moreover, it is common that a reasoner gets stuck after finding some justifications just to make sure there is no other justification.

Another possibility is reducing the search space for justifications. We can focus on a subset of the ontology to find where the problem of that subset is when combined with the rest. For example, we can ask for a justification \( J_T \) such that \( (J_T, A, R) \) is inconsistent. We can say that we are looking for justifications in taxonomy (TBox and RBox) and data (ABox) perspectives. Furthermore, the possibility can be also narrowed down further by choosing smaller subsets of the ontology. This is helpful if a user is confident that a part of the ontology is not the problem. In this case, the user can leave out the guessing for this part which leads to smaller search space.

### 3.2 Finding Justification under Fixed-Domain Semantics

We formalized the justifications framework for fixed-domain semantics. There are not many adjustment, except the entailment. Furthermore, we use term knowledge base for our definition. Note that the use of subset for minimality works fine because fixed-domain reasoning is monotonic.

**Definition 3.8** (Fixed-Justification). Let \( K \) be a knowledge base, and \( \Delta \) be a fixed-domain such that \( K \models_{\Delta} \eta \). \( J \) is a \( \Delta \)-justification for \( \eta \) in \( K \) if \( J \subseteq K \) and \( J \models_{\Delta} \eta \), and for all \( J' \subset J \), \( J' \not\models_{\Delta} \eta \).

We will use the term justification for fixed-justification if the context is clear. For any knowledge base \( K \) and an entailment \( \eta \), if \( K \models \eta \) then \( K \models_{\Delta} \eta \) for all fixed-domain \( \Delta \). The interesting question is whether it is also the case for justification, i.e., if \( J \) is a justification then it is also a fixed-justification for any \( \Delta \). This is not the case, since there probably exists a \( J' \subset J \) such that \( J' \not\models \eta \) but \( J' \models_{\Delta} \eta \).

We revisited our previous example but checking it in fixed-domain semantics. We let the domain be all individuals that can be found in the knowledge base.

**Example 3.9.** (AT Knowledge Base Inconsistency Fixed-Justification) Let \( \eta = \top \sqsubseteq \bot \) (represents inconsistency) and \( \Delta = \{ \text{whale, crocodile, goldfish, catfish} \} \). We obtain

\[
\begin{align*}
J_1 &= \{ \text{Mammal(whale), WaterAnimal(whale)\} \cup Mammal \subseteq \text{LandAnimal}, \text{LandAnimal} \subseteq \neg \text{WaterAnimal} \}. \\
J_2 &= \{ \text{Reptile} \sqsupseteq 4 \text{eat.Fish, Reptile(crocodile), Fish} \subseteq \neg \text{Reptile} \}. \\
J_3 &= \{ \text{Reptile} \sqsupseteq 4 \text{eat.Fish, Reptile(crocodile)} \} \cup \{ \text{Mammal(Whale), Mammal} \subseteq \neg \text{Fish} \}. \\
J_4 &= \{ \text{Reptile} \sqsupseteq 4 \text{eat.Fish, Reptile(crocodile), Fish} \subseteq \text{WaterAnimal} \} \cup \{ \text{Mammal(Whale), Mammal} \subseteq \text{LandAnimal, LandAnimal} \subseteq \neg \text{WaterAnimal} \}.
\end{align*}
\]

\( J_1, J_2, J_3 \) and \( J_4 \) are \( \Delta \)-justifications for \( \eta \) in \( K_{AT} \).
We get new justifications $J_2$ and $J_3$. The axiom $Reptile \sqsubseteq \geq 4 \text{eat.Fish}$ says a reptile eats more than four kind of fishes. Since there are disjointness axioms, then we only have two kind of fishes, goldfish and catfish. We already fixed the domain to four elements, then there is no more possible element that can fulfill the deficit. There are two justifications for this because each of them will decrease the number of possible Fish instances from four to three. The problem itself would not occur if there is no instance of $Reptile$.

In fact, we found the other justifications in the testing. Without disjointness between $Fish$ and $Mammal$, the inconsistency still occurs. It happens since we have whale as a $Mammal$, hence a $LandAnimal$. On the other hand, a $Fish$ must be a $WaterAnimal$, which is disjoint to $LandAnimal$. Consequently, it is still not possible to have whale as a $Fish$. This case is represented by justification $J_4$.

This example shows fixed-domain reasoning relies much on the domain size. Consider a new axiom $\alpha_1 = Reptile \sqsubseteq \geq 5 \text{eat.Fish}$ with the previous domain. Since there are only four individuals in the domain, the axiom would be always unsatisfiable regardless the other axioms. Thus, the singleton set $\{\alpha_1\}$ is a justification.

### 3.2.1 Black-box Algorithm

Recall that, due to Theorem 2.19, one can axiomatize a knowledge base $\mathcal{K}$ to $\mathcal{K} \cup \mathcal{FD}_\Delta$ to do the fixed-domain reasoning. Consequently, it is possible to use existing tools for finding justifications under standard semantics. Therefore, the complete steps are to axiomatize the original knowledge base and then to do the reasoning over the axiomatized knowledge base. Obviously, the axiomatization is not a costly task. Thus, the complexity of using black-box method depends heavily on the method complexity.

A doctoral thesis of Matthew Horridge consists of a comprehensive studies about justification [Hor11]. The concrete implementation of the thesis is the owl-explanation framework. Furthermore, the framework is expanded to OWL Explanation Workbench explanation-workbench which consists of a library and Protege plugin. The implementation itself used some optimizations, such as syntactic-locality-based modules, a structural expansion stage, a divide and conquer pruning strategy, and Hitting Set Tree (HST) optimizations [HPS09b]. The framework is an open-source software and available on GitHub.

This method is arguably the most accessible to users since the explanation-workbench is a default plug-in for Protegé. Thus, a user can axiomatize the knowledge base and use "Explain inconsistent ontology" as shown in Figure 3.1. The out-of-the-box usage for this feature uses Hermit as the reasoner. While explanation-workbench supports for finding laconic justifications [Hor11], in this work we focus on finding regular justifications. In short, a laconic justification contains only the relevant part of each axiom rather than a complete full axiom. Figure 3.2 shows the justification workbench window example. While it is very easy to use the Protegé justification plugin, we use the API for evaluating the black-box method. This approach gave us more controls that are not

[https://github.com/protegeproject/explanation-workbench](https://github.com/protegeproject/explanation-workbench)
provided by the plug-in.

![Figure 3.1: Explain inconsistent ontology feature in Protégé](image1)

![Figure 3.2: Explanations example in Protégé](image2)

We developed a glass-box algorithm for finding justifications under fixed-domain semantics. The approach is based on the translation from knowledge bases into ASP programs. We adapted some changes to naïve translation to suit our idea. We will discuss about this method the next chapter.
In this chapter, we represent a glass-box method for finding Δ-justifications. We modified the previous translation from SROIQ knowledge base into ASP programs. We constructed a new encoding such that the answer sets correspondence with the set of all justifications of the inconsistency.

4.1 Normalized Form

In this work, we consider the input knowledge base is in normalized form. While the original definition is based on the one used in HermiT [MSH09], there are slight differences in the normalized form used for fixed-domain semantics. Generally, this normalization is based on previous work [GRS16] unless otherwise noted.

We begin with preprocessing ABox such that it contains only individuals from the domain, except in nominal concepts. This can be obtained by model-preserving transformations ω(A) described in Table 4.1.

**Definition 4.1.** (Normalized Form) A GCI is normalized, if it is of the form ⊤ ⊑ ∪_{i=1}^n C_i, where C_i is of the form B, {a}, ∀r.B, ∃r.Self, ¬∃r.Self, ≥ n r.B, or ≤ n r.B, for B a literal concept, r a role, and n a positive integer. A TBox T is normalized, if each GCI in T is normalized. An ABox A is normalized if each concept assertion in A contains only a literal concept, each role assertion in A contains only an atomic role, and A contains at least one assertion. An RBox R is normalized, if each role inclusion axiom is of the form r ⊑ r’ or r_1 o r_2 ⊑ r’. A SROIQ knowledge base K = (T, A, R) is normalized if T, A, and R are normalized.

Similar with the normalized form definition, there are adjustments in the transformation. We can obtain a normalized form of the input knowledge base K by using the transformation Ω(K) as depicted in Table 4.2.
Table 4.2: Ω-Normalization of knowledge base axioms.

<table>
<thead>
<tr>
<th>Ω(Κ)</th>
<th>Ω(T ⊑ C ∪ C')</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ω(Κ) =</td>
<td>Ω(T ⊑ C ∪ C')</td>
</tr>
<tr>
<td>Ω(T ⊑ C ∪ C') = Ω(T ⊑ C ∪ C') ∪ Ω(T ⊑ ¬αC ∪ C1)</td>
<td></td>
</tr>
<tr>
<td>Ω(T ⊑ C ∪ C') = Ω(T ⊑ C ∪ C') ∪ Ω(T ⊑ ¬αC ∪ C1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>for C' of the form C' = C1 ∩ · · · ∩ Cn and n ≥ 2</td>
</tr>
</tbody>
</table>

Ω(T ⊑ C ∪ 3r.D) = Ω(T ⊑ C ∪ 1 r.D) |
Ω(T ⊑ C ∪ 4r.D) = Ω(T ⊑ C ∪ 4r.D) |
Ω(T ⊑ C ∪ n r.D) = Ω(T ⊑ C ∪ n r.D) |
Ω(T ⊑ C ∪ ≤ n r.D) = Ω(T ⊑ C ∪ ≤ n r.D) |
Ω(D(s)) = {αD(s)} ∪ Ω(T ⊑ ¬αD ∪ nnf(D)) |
Ω(r−(s,t)) = {r−(s,t)} |
Ω(r1 ◦ . . . ◦ r n ⊏ r) = {r1 ◦ r2 ⊏ r(r1 ◦ r2) } ∪ Ω(r1 ◦ r2 ◦ r3 ◦ . . . ◦ r n ⊏ r) |
Ω(β) = {β} for any other axiom β |

αC = \{ Qc if pos(C) = true \}
\{ ¬Qc if pos(C) = false \}

where Qc is a fresh concept name unique for C.

| pos(T) = false |
| pos(A) = true |
| pos(∃r.Self) = true |
| pos(C1 ∩ C2) = pos(C1) ∨ pos(C2) |
| pos(∀r.C1) = pos(C1) |
| pos(≥ n r.C1) = true |
| pos(≤ n r.C1) = false |

Note: A is an atomic concept, C(i) are arbitrary concept expressions, C is a possibly empty disjunction of concept expressions, D is a literal concept. The function ¬ is defined as ¬(¬A) = A and ¬(A) = ¬A for some atomic concept A.

4.2 Fixed-Domain Reasoning Encoding

While we can use standard semantics reasoner using Theorem 2.19, the ASP translation was introduced as an alternative method for fixed-domain reasoning. The main idea is to build an ASP program such that each answer set of the program coincides with the model under fixed-domain semantics. We start by recalling the naive encoding for SROIQ knowledge bases in a compact way [GRSI10].

Based on the paradigm, we have to choose what and how to guess and check. Naturally, the first thing that comes in mind is the interpretation. We generate the possible interpretation and then check whether it violates any axiom in the knowledge base or not. Note that it is possible because we have a fixed-domain, hence a bounded number.
of interpretations. Consider a knowledge base $\mathcal{K} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$, then we have

$$\Pi_{gen}(\mathcal{K}, \Delta) := \{
\forall X: \top(X), \not\top(X) \mid C \in NC(\mathcal{K}) \}
\cup
\{
\forall X: \top(X), \not\top(X) \mid C \in NC(\mathcal{K}) \}
\cup
\{
\forall r(X, Y): \top(X), \top(Y), \not\top(r(X, Y)) \mid r \in NR(\mathcal{K}) \}
\cup
\{
\forall r(X, Y): \top(X), \top(Y), \not\top(r(X, Y)) \mid r \in NR(\mathcal{K}) \}.
$$

(4.1)

(4.2)

(4.3)

(4.4)

(4.5)

(4.6)

(4.7)

(4.8)

(4.9)

It is quite intuitive that $\Pi_{gen}(\mathcal{K}, \Delta)$ spans all candidate interpretations based on the knowledge base signature. The next step is checking the existence of any axiom violation. We encode each part of the knowledge base: ABox, TBox, and RBox. The translation of the ABox $\mathcal{A}$ is straightforward. Each individual assertion $C(a)$ and role assertion $r(a, b)$ in the ABox is included in the program as a fact. Obvious contradictions ($C(X)$ and $\not C(X)$) are handled by semantics of strong negations in ASP. The same principle applies on the roles.

$$\Pi_{chk}(\mathcal{A}, \Delta) := \mathcal{A}
$$

(4.5)

Now consider the TBox $\mathcal{T}$ and remember that each GCI is of the form $\top \sqsubseteq \bigcup_{i=1}^{n} C_i$. We can say any individual should be a member of concept $C_i$ for any $i$. The axiom is violated if there is an individual which is not a member of any $C_i$, i.e., $\bigcap_{i=1}^{n} \not C_i \sqsubseteq \bot$. This can be translated directly into an ASP constraint.

$$\Pi_{chk}(\mathcal{T}, \Delta) := \{\forall \tau(C_1), ..., \tau(C_n), thing(X) : \top \sqsubseteq \bigcup_{i=1}^{n} C_i \text{ in } \mathcal{T} \}
$$

(4.6)

where $\tau(C)$ is a concept membership translation. We define $trans(C)$ for any form of (normalized) concept in the Table 4.3. Each $\tau(C)$ represents when an individual is not a member of any $C_i$, i.e., a member of $\not C_i$.

The last part is the RBox $\mathcal{R}$ which consist of role axioms of the form $r \sqsubseteq s$, $r_1 \circ r_2 \sqsubseteq s$, or $Dis(r, s)$. Similar with TBox encoding, we look for any violation of RBox axioms as constraints.

$$\Pi_{chk}(\mathcal{R}, \Delta) := \{\forall r(X, Y), not s(X, Y) : r \sqsubseteq s \in \mathcal{R} \} \cup
\{\forall r_1(X, Y), r_2(Y, Z), not s(X, Z) : r_1 \circ r_2 \sqsubseteq s \in \mathcal{R} \} \cup
\{\forall r(X, Y), s(X, Y) : Dis(r, x) \in \mathcal{R} \}.
$$

(4.7)

(4.8)

(4.9)

Now, we have a complete translation of axioms in knowledge base $\mathcal{K}$. These constraints and facts ensure any interpretation that violates any axiom is not an answer set of the program.

$$\Pi_{chk}(\mathcal{K}, \Delta) := \Pi_{chk}(\mathcal{T}, \Delta) \cup \Pi_{chk}(\mathcal{A}, \Delta) \cup \Pi_{chk}(\mathcal{R}, \Delta).
$$

(4.10)

Thus, we have a complete translation $\Pi(\mathcal{K}, \Delta)$. It is easy to see the complete program is based on the guess and check paradigm. $\Pi(\mathcal{K}, \Delta)$ will generate any possible interpretation with $\Pi_{gen}(\mathcal{K}, \Delta)$ and then check whether there exists any violated axiom or not.
<table>
<thead>
<tr>
<th>$C$</th>
<th>$\text{trans}(C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$\neg A(X)$</td>
</tr>
<tr>
<td>$\neg A$</td>
<td>$A(X)$</td>
</tr>
<tr>
<td>${a}$</td>
<td>${\neg O_a(X)}, {O_a(a)}$</td>
</tr>
<tr>
<td>$\forall r. A$</td>
<td>${\neg A(Y), ar(X,Y)}$</td>
</tr>
<tr>
<td>$\forall r. \neg A$</td>
<td>${A(Y), ar(X,Y)}$</td>
</tr>
<tr>
<td>$\exists r. \text{Self}$</td>
<td>$\neg ar(r, X, X)$</td>
</tr>
<tr>
<td>$\neg \exists r. \text{Self}$</td>
<td>$ar(r, X, X)$</td>
</tr>
<tr>
<td>$\geq n r. A$</td>
<td>$#\text{count} { Y, ar : r(X,Y), A(Y) } &lt; n$</td>
</tr>
<tr>
<td>$\geq n r. \neg A$</td>
<td>$#\text{count} { Y, ar : r(X,Y), \neg A(Y) } &lt; n$</td>
</tr>
<tr>
<td>$\leq n r. A$</td>
<td>$#\text{count} { Y, ar : r(X,Y), A(Y) } &gt; n$</td>
</tr>
<tr>
<td>$\leq n r. \neg A$</td>
<td>$#\text{count} { Y, ar : r(X,Y), \neg A(Y) } &gt; n$</td>
</tr>
</tbody>
</table>

**Note:** $O_a$ is a new concept name unique for a $ar(r, X, Y)$ is defined as follows:

$$ar(r, X, Y) := \begin{cases} r(X, Y) & \text{if } r \text{ is an atomic role} \\ s(Y, X) & \text{if } r \text{ is an inverse role and } r = s^{-1} \end{cases}$$

Table 4.3: Translation of Concept Expressions for naïve Encoding.

in $\Pi_{\text{chk}}(\mathcal{K}, \Delta)$. What are left must be the $\Delta$-models of the knowledge base $\mathcal{K}$.

$$\Pi(\mathcal{K}, \Delta) := \Pi_{\text{gen}}(\mathcal{K}, \Delta) \cup \Pi_{\text{chk}}(\mathcal{K}, \Delta) \quad (4.10)$$

Using this encoding, we have an ASP program that has correspondence with the original knowledge base. Each answer set is an interpretation that does not violate any axiom, i.e., the model of the knowledge base.

**Theorem 4.2.** Let $\mathcal{K}$ be a simplified SROIQ knowledge base, $I$ be an interpretation, $\Delta$ be a fixed-domain, and we define

$$B_I := \{ A(a) | a \in A^I, A \in N_C(\mathcal{K}) \cup \{ \top \} \} \cup \{ r(a, b) | (a, b) \in r^I r \in N_R(\mathcal{K}) \cup \{ a \} \}.$$ 

Then, $I \vDash_{\Delta} \mathcal{K}$ iff $B_I \in AS(\Pi(\mathcal{K}, \Delta))$.

Due to Theorem 4.2, we have some reasoning tasks that can be done naturally. Asking for answer sets of the program is asking for models of the knowledge base, hence model enumeration. It is also easy to do satisfiability testing as we can ask for a model. If the ASP solver says the encoding is unsatisfiable, then the knowledge base is inconsistent. However, further investigation has to be done for other standard reasoning problems such as subsumption, equivalence, and instance problem.
4.3 Inconsistency Justifications Encoding

We introduced an encoding for finding inconsistency justifications of an $SROIQ$ knowledge base, called debug encoding. For any normalized ontology $K = (T, A, R)$, the answer sets of $\Pi(K, \Delta)$ coincides with the inconsistency justifications. We introduce this translation as a glass-box algorithm for finding justifications under fixed-domain reasoning.

While the main idea is based on naïve encoding, we use the naff (negation-as-failure free) encoding as the base of the program. The naff translation is similar to naïve encoding, but it does not have any negation-as-failure and $\neg$-comparator. The differences will be self-explanatory in this encoding definitions.

We can divide the whole encoding into two parts. The first one is guessing axioms that represent choosing which subset of knowledge base we consider. This part uses activated atoms for each axiom. If we guess $\text{activated}(i)$ to be in the model, then we say $\alpha_i$ is in our subset. After guessing these activated atom and resolve the program, we will have a simplified program that coincide with a subset of the knowledge base. We will show this later using the splitting theorem [LT94].

The second part is checking the unsatisfiability of such computed subset. Normally, the idea is to check every axiom which is represented by constraints. In this way, we will have models if the program is satisfiable, and none otherwise. It is a bit tricky to do a one-shot computation using this approach. Therefore, we use a different technique for unsatisfiability checking. Instead of constraints, we use rules to detect any axiom violation that force an atom that represents inconsistency to be in the model. If we detect the existence of this “inconsistency” atom, then we make all of concept and role memberships follows. This expresses the principle of explosion. With this approach, any concept and role membership guessing that leads to axiom violation will have the same model, which is called “the unified model”.

We will have guessing, qualified number restriction, axioms encoding, and interpretation removal parts. Guessing and qualified number restriction parts ensure the generation of the candidate interpretations with consequent auxiliary predicates. Axioms encoding will detect if there is any axiom that is not satisfied by the guessed interpretation and force them to be the unified model. We remove the unified model using interpretation removal part. Until this point, we will have the subset of the knowledge base that inconsistent, but not necessary minimal. We appended them with answer sets preference encoding to ensure minimality which handled by asprin.

$$\Pi(K, \Delta) = \Pi_{gen}(K, \Delta) \cup \Pi_{nra}(K, \Delta) \cup \Pi_{chk}(K, \Delta) \cup \Pi_{inc} \cup \Pi_{pref}.$$ 

4.3.1 Candidate Generation

Let $K = (T, A, R)$ be a normalized $SROIQ$ knowledge base. Recall that, an interpretation $I$ is a tuple that consists a set of domain $\Delta^I$ and a function $\cdot^I$. The function $\cdot^I$ maps every concept to a subset of $\Delta^I$ and every role name to a subset of $\Delta^I \times \Delta^I$. Intuitively, $\Pi_{gen}$ spans all possible interpretations by guessing any possibility of concept
and role membership over the defined domain. We use disjunctive rules for this purpose. There are several other possibilities to do the guessing, but disjunctive is used as we will exploit the semantics later.

\[ \Pi_{gss}(K, \Delta) := \{ A(X), \neg A(X) := \text{thing}(X) \mid A \in N_C(K) \} \cup \{ r(X,Y), \neg r(X,Y) := \text{thing}(X), \text{thing}(Y) \mid r \in N_R(K) \} \cup \{ \text{thing}(a) \mid a \in \Delta \}. \] (4.11)

\[ \{ \text{incons} := A(X), \neg A(X) \mid A \in N_C(K) \} \cup \{ \text{incons} := r(X,Y), \neg r(X,Y) \mid r \in N_R(K) \}. \] (4.12)

Other thing to note is we avoid using negation-as-failure notation in this guessing part. One could argue that there is already strongly negated atoms in ASP which can be used to represent negative concept membership. The problem is a strongly negated atom and the positive counterpart cannot appear in the same answer set. It will not let us to use our idea about principle of explosion, which will be mentioned later.

We do not want any obvious contradiction occurring in a model. While \( \Pi_{gen}(K, \Delta) \) will not generate such case because disjunction minimality, it is still possible that some assertions cause this. Thus, we introduce a rule for each concept and role to catch if this happens.

\[ \Pi_{obv}(K, \Delta) := \{ \text{incons} := A(X), \neg A(X) \mid A \in N_C(K) \} \cup \{ \text{incons} := r(X,Y), \neg r(X,Y) \mid r \in N_R(K) \}. \] (4.13)

Finally, we introduce rules to represent the principle of explosion. Whenever an inconsistency is detected, all concept and role assertions follow. The rules unify all of inconsistent interpretations (w.r.t \( K \)) into one (classical) model. From this point forward, we call such a model \textit{unified-inconsistent} model.

\[ \Pi_{poe}(K, \Delta) := \{ A(X) := \text{incons} \mid A \in N_C(K) \} \cup \{ \neg A(X) := \text{incons} \mid A \in N_C(K) \} \cup \{ r(X,Y) := \text{incons} \mid r \in N_R(K) \} \cup \{ \neg r(X,Y) := \text{incons} \mid r \in N_R(K) \}. \] (4.14)

Thus, we have a complete part of the program to generate candidate interpretations \( \Pi_{gen}(K, \Delta) \). Furthermore, this part also ensures any inconsistent interpretation will be forced to be the unified model.

\[ \Pi_{gen}(K, \Delta) := \Pi_{gss}(K, \Delta) \cup \Pi_{obv}(K, \Delta) \cup \Pi_{poe}(K, \Delta). \] (4.15)

\subsection*{4.3.2 Qualified Number Restriction Encoding}

One of the reason why \textit{naive} encoding is not used as the base of this work is the usage of the \( \leq n \) operator. Consider the concept \( \leq n r.A \), the axiom restricts an individual on how many members of concept \( A \) it is connected through an r-role. For example, let \( a \) be an individual that needs to be checked whether it is a member of \( \leq n r.A \) or not. The intuitive way is counting how many individuals are \( r \)-connected from \( a \) and how many
of them are members of \( A \). However, this needs \(<\)-cardinality ASP rule which we do not want to have in the translation.

For the sake of simplicity, we define \( r.A(a) = \{ x \in A^I \mid (a,x) \in r^I \} \). Hence \( r.A(a) \) consists of all member of concept \( A \) that connected by role \( r \) from \( a \). The idea is to count individuals which are not a member of the concept where this restriction applies. There are two possibilities that an individual \( b \) is not in \( r.A(a) \): \((a,b) \notin r \) or \( b \notin A \). Furthermore, if \( b \) does not hold in any of both cases then \( b \) is an element in. Let \( n = |\Delta^I| \) and \( m = |\{ b \in \Delta^I \mid (a,b) \notin r.A \}| \). Hence, the number of individuals in \( r.A(a) \) is the \( n - m \). This is viable in fixed-domain semantics because there is a control to the number of all individuals.

**Proposition 4.3.** Let \( K \) be a SROIQ knowledge base, \( \Delta \) be a fixed-domain, and \( I \) is a \( \Delta \)-model of \( K \). Then \( (\geq n \ r.C)^I = \{ x \mid \#\{ y \in \Delta^I \mid y \notin C^I \text{ or } (x,y) \notin r^I \} < |\Delta^I| - n \} \).

**Proof.** Assume \( a \in (\geq n \ r.C)^I \). Then, there are at least \( n \) numbers of \( b \in \Delta^I \) such that \( b \in C^I \) and \((a,b) \in r^I \) by the standard semantics. Hence, there are less than \( |\Delta^I| - n \) elements \( b' \) do not satisfy \( b' \notin C^I \) and \((a,b') \notin r^I \). Then, there are less than \( |\Delta^I| - n \) such that \( b' \notin C^I \) or \((a,b') \notin r^I \).

And conversely for the other direction. \( \square \)

Hence, we can compute such relation between two individuals to be used later in the translation of axioms. A new auxiliary predicate is for each pair of concept (and its negation) and role. We define

\[
\Pi_{nra}(K) := \begin{array}{l}
\{ \text{not}_r.C(X,Y) :- \text{not}_r(X,Y) \mid C \in N_C(K), r \in N_R(K) \} \cup \quad (4.21) \\
\{ \text{not}_r.C(X,Y) :- \text{not}_C(Y) \mid C \in N_C(K), r \in N_R(K) \} \cup \quad (4.22) \\
\{ \text{not}_r\text{not}_r.C(X,Y) :- \text{not}_r(X,Y) \mid C \in N_C(K), r \in N_R(K) \} \cup \quad (4.23) \\
\{ \text{not}_r\text{not}_r.C(X,Y) :- C(Y) \mid C \in N_C(K), r \in N_R(K) \} \quad (4.24)
\end{array}
\]

\( \Pi_{nra}(K) \) does not change the interpretation built by \( \Pi_{gen}(K) \). It merely collects all individuals those satisfies previously mentioned conditions.

### 4.3.3 Axiom Encoding

We translate each axiom in the TBox, RBox, and ABox to check whether any of them is violated by the guessed interpretation or not. The main idea is to build a rule for each axiom that fires \textbf{incons} whenever the axiom is violated. The input knowledge base is in normalized form. We make sure atom \textbf{incons} holds whenever any constraint is violated instead of writing them as integrity constraints. The \textbf{incons} represents inconsistency occurs in corresponding interpretation candidate. If the program detects such case, then we ”build” a model where everything follows.
ABox Translation

First prune of the search space comes from ABox assertions. As the input is a normalized knowledge base, each concept assertion contains only a literal concept and each role assertion contains only an atomic role. These assertions are quite straightforward to encode. We define

\[
\Pi_{chk}(\mathcal{A}) := \{ A(a) : \neg \text{activated}(i) \mid A(a) \in \mathcal{A} \} \cup \{ \neg A(a) : \neg \text{activated}(i) \mid \neg A(a) \in \mathcal{A} \} \cup \{ r(a,b) : \neg \text{activated}(i) \mid r(a,b) \in \mathcal{A} \} \cup \{ \neg r(a,b) : \neg \text{activated}(i) \mid \neg r(a,b) \in \mathcal{A} \}. \tag{4.25}
\]

TBox Translation

Each axiom in the TBox is in the normalized form. The normalized form is a disjunction of concepts, i.e. each element in the domain should be member of at least one of these concepts. Hence, the violation occurs when we detect an individual that is not a member of any concept or equivalently a member of all negated concepts.

\[
\Pi_{chk}(\mathcal{T}) := \{ \text{incons} : \neg \text{activated}(i), \tau(C_1), \ldots, \tau(C_n), \text{thing}(X) \mid \top \subseteq \bigcup_{i=1}^{n} C_i \text{ in } \mathcal{T} \} \tag{4.29}
\]

We have a normalized knowledge base, thus we know the form of the concept \( C_i \). The translation \( \text{trans}(C_i) \) for each form of the concept is defined in Table 4.4. Each \( \text{trans}(C_i) \) represents membership checking of concept \( \neg C_i \). If there exists any individual which is a member of all \( \neg C_i \), then the axiom is not satisfied and \text{incons} is fired.

RBox Translation

Similar with TBox, the input RBox should be in normalized form. Then for each axiom in RBox \( \mathcal{R} \), we have either a (simplified) role chain, disjointness or role inclusion axiom. We translated RBox axioms in a similar way with TBox, that we detect if there are individuals violate the axiom.

\[
\Pi_{chk}(\mathcal{R}) := \{ \text{incons} : \neg r(X,Y), s(X,Y). \mid \text{Dis}(r,s) \in \mathcal{R} \} \cup \{ \text{incons} : \neg r(X,Y), \neg s(X,Y). \mid r \subseteq s \in \mathcal{R} \} \cup \{ \text{incons} : s_1(X,Y), s_2(Y,Z), \neg r(X,Z). \mid s_1 \circ s_2 \subseteq r \in \mathcal{R} \}. \tag{4.30}
\]

Removing Unwanted Models

We simply remove the models without \text{incons} using a constraint. These (classical) models represent the models of the original knowledge base. Thus, what is left as the
4.3 Inconsistency Justifications Encoding

<table>
<thead>
<tr>
<th>C</th>
<th>trans(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>not_A(X)</td>
</tr>
<tr>
<td>¬A</td>
<td>A(X)</td>
</tr>
<tr>
<td>{a}</td>
<td>{not_O(a)(X)}, {O(a)}</td>
</tr>
<tr>
<td>\forall r.A</td>
<td>{not_A(Y), ar(X,Y)}</td>
</tr>
<tr>
<td>\forall r.¬A</td>
<td>{A(Y), ar(X,Y)}</td>
</tr>
<tr>
<td>\exists r.Self</td>
<td>not_ar(r, X, X)</td>
</tr>
<tr>
<td>¬\exists r.Self</td>
<td>ar(r, X, X)</td>
</tr>
<tr>
<td>≥ n r.A</td>
<td>#count{Y, ar : not_ar_A(X, Y)} &gt; (m − n)</td>
</tr>
<tr>
<td>≥ n r.¬A</td>
<td>#count{Y, ar : not_ar_not_A(X, Y)} &gt; (m − n)</td>
</tr>
<tr>
<td>≤ n r.A</td>
<td>#count{Y, ar : r(X, Y), A(Y)} &gt; n</td>
</tr>
<tr>
<td>≤ n r.¬A</td>
<td>#count{Y, ar : r(X, Y), not_A(Y)} &gt; n</td>
</tr>
</tbody>
</table>

Note: \(O_a\) is a new concept name unique for a \(ar(r, X, Y)\) is defined as follows:

\[
ar(r, X, Y) := \begin{cases} 
  r(X, Y) & \text{if } r \text{ is an atomic role} \\
  s(Y, X) & \text{if } r \text{ is an inverse role and } r = s^-
\end{cases}
\]

\[m = |\Delta|^2\]

Table 4.4: Translation of Concept Expressions for debug Encoding.

The answer set should be the unified inconsistent model if it is a minimal model.

\[
\Pi_{inc} := \{ :- \text{ not incons.}\}
\]

Preferences and asprin

So far we have an encoding such that if the correspondence subset knowledge base is inconsistent, there is an answer set. We still missed the minimality aspect for justification. Hence, we have to find preferred answer sets that is minimal in the term of activated atoms. Getting preferred answer sets sometimes is tricky. Due to that, a system called asprin [BDRS15] (stands ASP for preferences handling) was designed. Using asprin, users can declare preferences for answer sets and compute an ASP program based on them. It also comes with several pre-defined preferences in the library.

Here is an example of writing a preference in asprin:

\[
\#\text{preference} \{\text{prices, less(weight)}\}\{40: \text{bread}, 70: \text{meat}, 20: \text{milk}\}
\]

The first argument, prices, is the name of the preference. The type is defined by the second argument, less(weight). As the name suggests, this type of preference will look for answer sets with the atoms value as small as possible with respect of assigned value. For example, an answer set without any bread, meat, or milk will have a value 0, which is the minimal value possible, hence the most preferred. An answer set with bread and milk with the value 60 is preferred to an answer set with meat only with the value 70.
Since it is a preference, it is also possible to have all of them in the answer set if there is no answer set with smaller value possible.

One of the pre-defined preference is the subset relation. Thus, we can get minimal answer sets with the respect of the subset relation on a defined set. It reminds us of the minimality requirement in the justification framework. Since we already defined a program that computes all subsets that satisfy a certain entailment, we let asprin handle the minimality aspect.

This part will find the minimal subsets for the justifications. It uses asprin syntax, so having clingo in the system is not enough. It is possible to remove this part, and get “non-minimal justifications”. This part will guess which activated(i) atoms are active. Each activated(i) correspond to an axiom in the knowledge base. If activated(i) is guessed to be false, then the axiom αi is not represented in the program anymore. We define Π_{pref} as the code below.

\[
\begin{align*}
\text{:- not incons.} \\
\{\text{activated}(X) :- X = 0..n}. \\
\text{#optimize(p).} \\
\text{#preference(p, subset)}\{ \\
\quad \text{activated}(C) : C=0..n}. \\
\}
\end{align*}
\]

where \( n \) is the number of axioms.
5 Soundness and Completeness

In this chapter, we will show that the translation works as what we expected. We divide the program into two parts using the splitting theorem. We show that the first part of the program does the subset guessing of the knowledge base. The simplified program using the answer set of the first part corresponds to an encoding of checking satisfiability of such knowledge base subset.

5.1 Splitting Theorem

Explaining the expected result of an ASP program is a quite complex task. We use the splitting theorem and thereby partition the logic program to focus on the behavior of each part. The splitting theorem makes use of splitting set to cut the program.

**Definition 5.1.** (Splitting Set[LT94]) A splitting set of a program $\Pi$ is a set $U$ of literals such that, for every rule $r \in \Pi$, if $\text{head}(r) \cap U \neq \emptyset$ then $\text{lit}(r) \subseteq U$. We call the part of the program which consists of rules $r \in \Pi$ such that $\text{lit}(r) \subseteq U$ as $b_U(\Pi)$ (stands for bottom), and $\Pi \setminus b_U(\Pi)$ as $t_U(\Pi)$ (stands for top).

Intuitively, one can use this splitting set to split a program into two parts and compute the answer sets in two phases. The first phase is computing the answer set of $b_U(\Pi)$. We can use each answer set $X$ of $b_U(\Pi)$ for simplifying $t_U(\Pi)$.

**Definition 5.2.** (Splitting Set Solution[LT94]) Consider two sets of literals $U$, $X$, and a program $\Pi$. We define a program $e_U(\Pi, X)$ which consists of all rules $r'$ built from each $r \in \Pi$ such that $\text{pos}(r) \cap U$ is a part of $X$ and $\text{neg}(r) \cap U$ is disjoint from $X$. Each $r'$ itself is defined by $\text{head}(r') = \text{head}(r)$, $\text{pos}(r') = \text{pos}(r) \setminus U$, $\text{neg}(r') = \text{neg}(r) \setminus U$. Let $U$ be a splitting set of a program $\Pi$. A solution to $\Pi$ w.r.t $U$ is a pair $(X, Y)$ of sets of literals such that

- $X$ is an answer set for $b_U(\Pi)$,
- $Y$ is an answer set for $e_U(t_U(\Pi), X)$,
- $X \cup Y$ is consistent.

Informally, we get $e_U(t_U(\Pi), X)$ in two steps for any rule. First, remove all atoms $A$ if $A \in X$ and not $B$ if $B \in U \setminus X$ from the body of the rule. Then, remove any rule which still contains any atom from $U$.

**Theorem 5.3.** Let $U$ be a splitting set for program $\Pi$. A set $A$ of literals is a consistent answer set for $\Pi$ if and only if $A = X \cup Y$ for some solution $(X, Y)$ to $\Pi$ with respect to $U$. 

33
5 Soundness and Completeness

5.2 Axiom Guessing Correctness

The idea is to split the program by means of the activated atom. Then, we show that the top part of the program does the knowledge base subset guessing, while the rest does the inconsistency checking.

**Proposition 5.4.** Let $\mathcal{K}$ be a knowledge base, $\Delta$ be a fixed-domain, and $U(\Pi(\mathcal{K}, \Delta)) = \{\text{activated}(i) \mid \text{activated}(i) \in \Pi(\mathcal{K}, \Delta)\}$. Then, $U(\Pi(\mathcal{K}, \Delta))$ is a splitting set of $\Pi(\mathcal{K}, \Delta)$, where

(i) $b_U(\Pi(\mathcal{K}, \Delta)) = \Pi_{\text{act}}(\mathcal{K}, \Delta)$, and

(ii) $t_U(\Pi(\mathcal{K}, \Delta)) = \Pi_{\text{gen}}(\mathcal{K}, \Delta) \cup \Pi_{\text{nra}}(\mathcal{K}, \Delta) \cup \Pi_{\text{chk}}(\mathcal{K}, \Delta) \cup \Pi_{\text{inc}}$.

**Proof.** It is easy to see that $U(\Pi(\mathcal{K}, \Delta))$ is a splitting set of $\Pi(\mathcal{K}, \Delta)$ because activated($X$) atoms only appear in $\Pi_{\text{act}}(\mathcal{K}, \Delta)$, thus no other atom needs to be included in $U(\Pi(\mathcal{K}, \Delta))$.

(i) For each rule in $\Pi_{\text{act}}(\mathcal{K}, \Delta)$, we only have activated atoms, thus satisfies $\text{lit}(r) \subseteq U$. Meanwhile, there is no other part of the program $\Pi(\mathcal{K}, \Delta)$ that has any rule with activated atoms in the head. Thus, $b_U(\Pi(\mathcal{K}, \Delta))$ only consists of rules in $\Pi_{\text{act}}(\mathcal{K}, \Delta)$.

(ii) Follows immediately from the definition $t_U(\Pi) = \Pi \setminus b_U(\Pi)$.

One may argue that there should be more atoms in $U(\Pi(\mathcal{K}, \Delta))$ because of the translation of choice rules to normal rules. However, it will not change the program splitting and never be included in the answer sets. For the sake of simplicity, $U(\Pi(\mathcal{K}, \Delta))$ is as defined previously. Recall that we only have the activated guessing rule in $b_U(\Pi(\mathcal{K}, \Delta))$, and the splitting theorem allows to solve $b_U(\Pi(\mathcal{K}, \Delta))$ first. If we have an ontology with size $n$, then we have $2^n$ answer sets of $b_U(\Pi(\mathcal{K}, \Delta))$ naturally from the fact that, it represents the power set of $U(\Pi(\mathcal{K}, \Delta))$. This part represents the guessing of justification candidates.

**Proposition 5.5.** For any SROIQ knowledge base $\mathcal{K}$ and any fixed-domain $\Delta$, it holds that $\text{AS}(b_U(\Pi(\mathcal{K}, \Delta))) = 2^{U(\Pi(\mathcal{K}, \Delta))}$

**Proof.** Follows immediately from the semantics of the choice rule.

Recall that we consider a knowledge base $\mathcal{K}$ as a sequence of axioms $\alpha_i$. We introduce a notation to keep the link between activated atoms guessing and corresponding subset of the knowledge base. We simply keep the connection of both $i$ in activated($i$) and $\alpha_i$.

**Definition 5.6.** Let $X \in 2^{U(\Pi(\mathcal{K}, \Delta))}$, we define $\mathcal{K}^X = \{\alpha_i \in \mathcal{K} \mid \text{activated}(i) \in X\}$.

The definition gives a correspondence between set $X$ to a subset of $\mathcal{K}$. Obviously this is a one-to-one correspondence where for each $X \in 2^{U(\Pi(\mathcal{K}, \Delta))}$, $\mathcal{K}^X$ represents a subset of $\mathcal{K}$ and vice versa.
Example 5.7. Let $\mathcal{K}$ be a knowledge base where $\mathcal{K} = \{\alpha_1, \alpha_2\}$, and

- $X_0 = \emptyset$. Then, we have $\mathcal{K}^{X_0} = \emptyset$.
- $X_1 = \{\text{activated}(1)\}$. Then, we have $\mathcal{K}^{X_1} = \{\alpha_1\}$.
- $X_2 = \{\text{activated}(2)\}$. Then, we have $\mathcal{K}^{X_2} = \{\alpha_2\}$.
- $X_3 = \{\text{activated}(1), \text{activated}(2)\}$. Then, we have $\mathcal{K}^{X_3} = \{\alpha_1, \alpha_2\}$.

Thus, it is enough to prove that the remaining program $e_U(\Pi(\mathcal{K}, \Delta), X)$, is an encoding for checking consistency of $\mathcal{K}^X$, i.e., in this approach: $e_U(\Pi(\mathcal{K}, \Delta), X)$ has an answer set iff $\mathcal{K}^X$ is inconsistent. For the sake of simplicity, we represent the program $e_U(\Pi(\mathcal{K}, \Delta), X)$ as $\Pi'(\mathcal{K}^X, \Delta)$. We could say that $\Pi'(\mathcal{K}^X, \Delta)$ is a translation for $\Delta$-inconsistency checking of a knowledge base $\mathcal{K}^X$.

Definition 5.8. Let $\mathcal{K} = (T, A, R)$ be a SROIQ knowledge base.

$$\Pi'_{\text{chk}}(T) := \{\text{incons} := \text{trans}(C_1), ..., \text{trans}(C_n), \text{thing}(X) \mid T \subseteq \bigsqcup_{i=1}^n C_i \text{ in } T\}$$

$$\Pi'_{\text{chk}}(A) := \{A(X) \mid A(X) \in A\} \cup \{\text{not}_a(A(X)) \mid \lnot A(X) \in A\} \cup \{r(X,Y) \mid r(X,Y) \in A\} \cup \{\text{not}_r(X,Y) \mid \lnot r(X,Y) \in A\}$$

$$\Pi'_{\text{chk}}(R) := \{\text{incons} := r(X,Y), \text{not}_s(X,Y) \mid r \subseteq s \in R\} \cup \{\text{incons} := r(X,Y), s(X,Y) \mid \text{Dis}(r,s) \in R\} \cup \{\text{incons} := s_1(X,Y), s_2(Y,Z), \text{not}_r(X,Z) \mid s_1 \circ s_2 \subseteq r \in R\}.$$}

$$\Pi'_{\text{chk}}(K) := \Pi'_{\text{chk}}(T) \cup \Pi'_{\text{chk}}(A) \cup \Pi'_{\text{chk}}(R).$$

Lemma 5.9. Let $\mathcal{K}$ be a knowledge base, $\Delta$ be a fixed-domain, and $X \in 2^{U(\Pi(\mathcal{K}, \Delta))}$. We define $\Pi'(\mathcal{K}^X, \Delta) = \Pi_{\text{gen}}(\mathcal{K}, \Delta) \cup \Pi'_{\text{chk}}(\mathcal{K}^X) \cup \Pi_{\text{inc}}$. It holds that $e_U(\Pi(\mathcal{K}, \Delta), X) = \Pi'(\mathcal{K}^X, \Delta)$.

Proof. Recall that $b_U(\Pi(\mathcal{K}, \Delta))$ is $\Pi_{\text{gen}}(\mathcal{K}, \Delta) \cup \Pi'_{\text{chk}}(\mathcal{K}) \cup \Pi_{\text{inc}}$. We know that in $\Pi_{\text{gen}}(\mathcal{K}, \Delta) \cup \Pi_{\text{inc}}$ part, there is no atom $\text{activated}$. Then, each rule is taken as it is, i.e., $\Pi_{\text{gen}}(\mathcal{K}, \Delta) \cup \Pi_{\text{inc}} \subseteq e_U(\Pi(\mathcal{K}, \Delta), X)$. Now, we show $e_U(\Pi'_{\text{chk}}(\mathcal{K}), X) = \Pi'_{\text{chk}}(\mathcal{K}^X)$. Any rule in $\Pi'_{\text{chk}}(\mathcal{K})$ is of the form $\Pi'_{\text{chk}}(\alpha_i)$ for any $\alpha_i \in T \cup A \cup R$ and $\text{activated}(i) \in X$. $e_U(\Pi'_{\text{chk}}(\alpha_i), X) \in \Pi'_{\text{chk}}(\mathcal{K}^X)$. Let $\alpha_i$ be a GCI of the form $\top \subseteq \bigsqcup_{i=1}^n C_i$ and $\text{activated}(i) \in X$. Thus,

- $\text{incons} := \text{activated}(i), \text{trans}(C_1), ..., \text{trans}(C_n), \text{thing}(X) \in \Pi(\mathcal{K}, \Delta),$
- $e_U(\Pi'_{\text{chk}}(\alpha_i), X) = \text{incons} := \text{trans}(C_1), ..., \text{trans}(C_n), \text{thing}(X).$

We know that $\alpha_i \in \mathcal{K}^X$, thus $\text{incons} := \text{trans}(C_1), ..., \text{trans}(C_n), \text{thing}(X) \in \Pi'_{\text{chk}}(\mathcal{K}^X)$. All remaining cases can be treated analogously. □
5 Soundness and Completeness

5.3 Inconsistency Checking Correctness

Now, we can focus on proving that $\Pi'(K^X, \Delta)$ will give the answer sets as we expected. First, we define a model whenever any inconsistency occurs called unified inconsistent model. A unified inconsistent model consists of all concept memberships, role relations, and auxiliary atoms in $\Pi'(K^X, \Delta)$.

**Definition 5.10.** (Unified Inconsistent Model) Let $K$ be a SROIQ knowledge base, and $\Delta$ be a fixed-domain. The unified inconsistent model $M^U(K, \Delta)$ is a classical interpretation of $\Pi(K, \Delta)$, where

- $\text{thing}(a) \in M^U(K, \Delta)$ for each $a \in \Delta$,
- $\{C(a), \text{not}_C(a)\} \subseteq M^U(K, \Delta)$ for each $C \in N_C(K)$, $a \in \Delta$,
- $\{r(a, b), \text{not}_r(a, b)\} \subseteq M^U(K, \Delta)$ for each $r \in N_R(K)$, $a, b \in \Delta$,
- $\{\text{not}_r.C(a, b), \text{not}_r.\text{not}_C(a, b)\} \subseteq M^U(K, \Delta)$ for each $C \in N_C(K)$, $a \in \Delta$,
- $\text{incons} \in M^U(K, \Delta)$
- there is no other atom in $M^U(K, \Delta)$

**Lemma 5.11.** Let $K$ be a SROIQ knowledge base, and $\Delta$ be a fixed-domain. $M^U(K, \Delta)$ is a classical model of $\Pi'(K^X, \Delta)$.

**Proof.** We show $M^U(K, \Delta)$ is a model of each rule in $\Pi'(K^X, \Delta)$. Recall that $\Pi'(K^X, \Delta) = \Pi_{gen}(K, \Delta) \cup \Pi'_{chk}(K^X) \cup \Pi_{inc}$. For each rule $\rho \in \Pi'(K^X, \Delta)$,

- $\rho \in \Pi_{gen}(K, \Delta)$. Then, $\rho$ is of the form:
  - $\text{A}(X), \text{not}_A(X) : \text{thing}(X)$. Since $\{\text{A}(X), \text{not}_A(X)\} \cap M^U(K, \Delta) \neq \emptyset$ for any $\text{grnd}(R)$, $M^U(K, \Delta)$ is a model of $\rho$.
  - $r(X, Y), \text{not}_r(X, Y) : \text{thing}(X), \text{thing}(Y)$. Since $\{r(X, Y), \text{not}_r(X, Y)\} \cap M^U(K, \Delta) \neq \emptyset$ for any $\text{grnd}(R)$, $M^U(K, \Delta)$ is a model of $\rho$.
  - $\text{thing}(X)$. Since $\{\text{thing}(X)\} \cap M^U(K, \Delta) \neq \emptyset$ for any $\text{grnd}(R)$, $M^U(K, \Delta)$ is a model of $\rho$.

- $\rho \in \Pi'_{chk}(K^X)$. Then, there are two cases:
  - $\rho$ is a translation of an axiom $\alpha_i$. Thus, we know $\text{head}(r) = \{\text{incons}\}$. Since $\{\text{incons}\} \subseteq M^U(K, \Delta) \neq \emptyset$, $M^U(K, \Delta)$ is a model of $\rho$.
  - $\rho$ is a translation of an assertion. Then, $\rho$ is of the form $A(X), \text{not}_A(X), r(X, Y), \text{not}_r(X, Y)$. Since $\text{head}(\rho) \cap M^U(K, \Delta) \neq \emptyset$, $M^U(K, \Delta)$ is a model of $\rho$.

- $\rho \in \Pi_{inc}$. Then, $\rho$ is $:\text{not}_\text{incons}$. We know that $\text{incons} \in M^U(K, \Delta)$, consequently $M^U(K, \Delta)$ satisfies the constraint.
5.3 Inconsistency Checking Correctness

Note that Lemma 5.11 does not imply $M^U(\mathcal{K}, \Delta)$ is an answer set unless it is a minimal model. The unified inconsistent model represents anything follows from contradiction. Furthermore, we linked such a model with the atom \textit{incons}.

**Lemma 5.12.** Let $\mathcal{K}$ be a knowledge base, $\Delta$ be a fixed-domain, $a, b \in \Delta$ and $M$ is a model of $\Pi'(\mathcal{K}^X, \Delta)$. $\textit{incons} \in M$ iff $M = M^U(\mathcal{K}, \Delta)$.

**Proof.** $\Rightarrow$ assume $\textit{incons} \in M$. Then,

- $\text{thing}(a) \in M$ because $\text{thing}(a) \in \Pi'(\mathcal{K}^X, \Delta)$,
- $\{C(a), \text{not}_C(a)\} \subseteq M$ because $C(a) :- \textit{incons} \in \Pi'(\mathcal{K}^X, \Delta)$, and $\text{not}_C(a) :- \textit{incons} \in \Pi'(\mathcal{K}^X, \Delta)$
- $\{r(a, b), \text{not}_r(a, b)\} \subseteq M$ because $r(a, b) :- \textit{incons} \in \Pi'(\mathcal{K}^X, \Delta)$, and $\text{not}_r(a, b) :- \textit{incons} \in \Pi'(\mathcal{K}^X, \Delta)$
- $\{\text{not}_r.C(a, b), \text{not}_r.\text{not}_C(a, b)\}$ because $\text{not}_r.C(a, b) :- \textit{incons} \in \Pi'(\mathcal{K}^X, \Delta)$, and $\text{not}_r.\text{not}_C(a, b) :- \textit{incons} \in \Pi'(\mathcal{K}^X, \Delta)$
- $\textit{incons} \in M^U(\mathcal{K}, \Delta)$ because of the assumption,
- there is no other atom in $M$ because there is no other predicate exists in $\Pi(\mathcal{K}, \Delta)$.

$\Leftarrow$ assume $M = M^U(\mathcal{K}, \Delta)$, then $\textit{incons} \in M$ because of the definition.

Recall that the semantics of constraints is preventing unwanted models. Thus, we temporarily leave out $\Pi_{nc}$ and focus on the other parts. We denote $\Pi''(\mathcal{K}^X, \Delta) = \Pi_{gen}(\mathcal{K}, \Delta) \cup \Pi'_{chk}(\mathcal{K}^X)$. Obviously, $M$ is a model of $\Pi'(\mathcal{K}^X, \Delta)$ iff it is a model of $\Pi''(\mathcal{K}^X, \Delta)$ and does not violate $\Pi_{nc}$.

Other classical models of $\Pi''(\mathcal{K}^X, \Delta)$ are expected to represent constructible interpretations. The first thing to check is the guessing part for concepts and roles, i.e., $\Pi_{gen}(\mathcal{K}, \Delta)$. We expect this part spans all possible interpretation candidates. Consider a rule (before grounded): the code below.

\[
\text{C}(X), \text{not}_C(X) :- \text{thing}(X).
\]

$M$ is a model of the rule iff $M$ consists of $C(X)$ or $\text{not}_C(X)$ or both. The case $\{C(X), \text{not}_C(X)\} \subseteq M$ imposes the existence of atom $\textit{incons}$ since there exists the rule $\textit{incons} :- C(X), \text{not}_C(X)$. Due to Lemma 5.12, the case only happens in the
unified inconsistent model. The two other cases represent individual assertions which induce interpretations.

Notice that for each fixed-domain interpretations $\mathcal{I} = (\Delta^I, \mathcal{I})$, there is an ABox $\mathcal{A}_I := \{r(a, b) | (a, b) \in r^I \cup \{A(a) | a \in A^I\}$. Conversely, every constructible ABox $\mathcal{A}$ in that way has a correspondence interpretation $\mathcal{I}_A$.

**Proposition 5.13.** Let $\mathcal{B}(\mathcal{K}, \Delta)$ be the set of all constructible ABox interpretations and $\mathcal{M}$ be the set of all classical models of $\Pi_{\text{gen}}(\mathcal{K}, \Delta)$, then $\mathcal{M} = \mathcal{B}(\mathcal{K}, \Delta) \cup M_U(\mathcal{K}, \Delta)$.

Consequently, any answer set is either an interpretation or an unified inconsistent model. Note that an answer set consists of not only assertions, but several auxiliary predicates. Thus, we define operator $\beta$ to extract the assertions from an answer set.

**Definition 5.14.** Let $\mathcal{K}$ be a SROIQ knowledge base, $\Delta$ be a fixed-domain and $M \in \mathcal{A}(\Pi(\mathcal{K}, \Delta))$. We define

$$\beta(\mathcal{K}, \Delta)(M) = \{\neg C(a), C(a) \in M | C \in N_C(\mathcal{K})\} \cup \{\neg r(a, b), r(a, b) \in M | r \in N_R(\mathcal{K})\}.$$

The question is whether there are more than one answer sets represent each constructible assertions. We show that each $B \in \mathcal{B}(\mathcal{K}, \Delta)$ is represented by only one model (of rules) by the nature of the translation.

**Proposition 5.15.** Let $\mathcal{K}$ be a SROIQ knowledge base, $\Delta$ be a fixed-domain. For each $B \in \mathcal{B}(\mathcal{K}, \Delta)$, there is at most one $M$ such that $M$ is a model of $\Pi'(\mathcal{K}, \Delta)$ and $\beta(\mathcal{K}, \Delta)(M) = B$.

**Proof.** Recall that $\Pi'(\mathcal{K}^X, \Delta) = \Pi_{\text{gen}}(\mathcal{K}, \Delta) \cup \Pi_{\text{chk}}'(\mathcal{K}^X) \cup \Pi_{\text{inc}}$. Thus, we need to check $\Pi_{\text{chk}}'(\mathcal{K}^X) \cup \Pi_{\text{inc}}$ part. It is easy to see that $\Pi_{\text{chk}}'(\mathcal{K}^X)$ contains only positive rules, hence deterministic and $\Pi_{\text{inc}}$ consists of a constraint, hence only removing models.

Due to this claim, we have a correspondence between the classical models of interpretation generation program part and the interpretations of the knowledge base. What remains is to show that corresponding classical model for any non-model interpretation is not a model of the program due to the constraints in $\Pi_{\text{chk}}'(\mathcal{K}^X)$.

**Definition 5.16.** Let $B_U$ be the set of all ground atoms, and $U$ be the set of all constructible ABox interpretations. Then, $\mathcal{I} \subseteq 2^{B_U}$ is a collection of sets of ground atoms, and $\mathcal{B} \subseteq 2^U$ is a collection of sets of ABox assertions. We say $\mathcal{I}$ and $\mathcal{B}$ correspond to each other, in symbols $\mathcal{I} \equiv \mathcal{B}$ iff

1. for each $I \in \mathcal{I}$, it holds that $\beta(\mathcal{K})(I) \in \mathcal{B}$
2. for each $B \in \mathcal{B}$, there exists $I \in \mathcal{I}$ such that $\beta(\mathcal{K})(I) = B$
3. $|I| = |\mathcal{B}|$
5.3 Inconsistency Checking Correctness

Using the notion of correspondence, we show that $\Pi''(K^X, \Delta)$ computes exactly $\Delta$-models of $K^X$ and it is possible to extract the model from any computed answer set of $\Pi''(K)$.

Lemma 5.17. For any $M \in AS(\Pi''(K^X, \Delta)) \setminus \{M^U(K, \Delta)\}$ and $B = \beta_K(M)$, then $B \in B_K$ and $I_B \models K$.

Proof. We show $B \in B_K$ and $I_B \models K$. $B \in B_K$ follows immediately from Proposition 5.13. We show the interpretation $I_B \models K^X$, i.e., $I_B \models \alpha$ for each $\alpha \in K^X$. Assume, for the sake of contradiction $I_B \not\models \alpha$ for some $\alpha \in K^X$. There are three possible cases:

- $\alpha \in R$. Then, $\alpha$ is of the form:

  - $\alpha = r \subseteq s$. Then, there exists a pair $(a, b)$, where $a, b \in \Delta^I_B$, $(a, b) \in r^I_B$ and $(a, b) \notin s^I_B$. By the definition of program $\Pi'(K^X, \Delta)$, there exists rule $\text{incons} \Leftarrow r(X,Y), \text{not.s}(X,Y)$. Consequently, $\text{incons} \in M$ and contradicts Lemma 5.12 $I_B \models r \subseteq s$.

  - $\alpha = \text{Dis}(r, s)$. Then, there exists a pair $(a, b)$ where $a, b \in \Delta^I_B$, $(a, b) \in r^I_B$ and $(a, b) \notin s^I_B$. By the definition of program $\Pi'(K^X, \Delta)$, there exists rule $\text{incons} \Leftarrow r(X,Y), s(X,Y)$. Thus, $\text{incons} \in M$ and contradicts Lemma 5.12 $I_B \models \text{Dis}(r, s)$.

  - $\alpha = r_1 \circ r_2 \subseteq s$. Hence, there are three individuals $a, b, c \subseteq \Delta^I_B$ where $(a, b) \in r_1^{I_B}$, $(b, c) \in r_2^{I_B}$ but $(a, c) \notin s^{I_B}$. By definition of program $\Pi'(K^X, \Delta)$, there exists rule $\text{incons} \Leftarrow r_1(X,Y), r_2(Y,Z), \text{not.s}(X,Z)$. Consequently, $\text{incons} \in M$ and contradicts Lemma 5.12 $I_B \models r_1 \circ r_2 \subseteq s$.

- $\alpha \in T$. Then, $\alpha$ is of the form $\top \subseteq \bigcup_{i=1}^n C_i$. Since $I_B \not\models \alpha$, then there exists some individual $a$ where $a \notin C_i^{I_B}$ for all $1 \leq i \leq n$. By the definition of $\Pi'(K^X, \Delta)$, there exists rule $\text{incons} \Leftarrow \text{trans}(C_1), ..., \text{trans}(C_n), \text{thing}(X)$. We know there is such $\text{trans}(C_i)$ that is not satisfied for any grounded instances of the rule because otherwise $\text{incons}$ will be fired and yield a contradiction. We distinguish:

  - $C_i = A$, then $\text{trans}(C_i) = \text{not.a}(X)$ and $A(a) \in B$. Consequently $a \in A^{I_B}$, which contradicts the assumption $I_B \not\models C_i$.

  - $C_i = \neg A$, then $\text{trans}(C_i) = A(X)$ and $\text{not.a}(a) \in B$. Consequently $a \in \neg A^{I_B}$, which contradicts the assumption $I_B \not\models C_i$.

  - $C_i = \{a\}$, then $\text{trans}(C_i) = \text{not.Oa}(X)$ and $\text{Oa}(a) \in B$, thus necessarily $\text{Oa}(a) \in B$. Consequently we have $a \in O_a^{I_B}$ with $O_a$ as nominal guard concept, and therefore $\{a\}^{I_B} = \{a\}$, which contradicts the assumption.

  - $C_i = \exists r.\text{Self}$, then $\text{trans}(C_i) = \text{not.r}(X, X)$ and $r(a, a) \in B$. Consequently, $(a, a) \in r^{I_B}$ which contradicts the assumption $I_B \not\models C_i$.

  - $C_i = \forall r.A$, then $\text{trans}(C_i) = \{r(X,Y), \text{not.a}(Y)\}$ and $A(b) \in B$ whenever $r(a, b) \in B$. Consequently, $(a, b) \in r^{I_B}$ implies $b \in A^{I_B}$, which contradicts the assumption $I_B \not\models C_i$.
\textbf{5 Soundness and Completeness}

- $C_i \supseteq n \cdot r.A$, then $\text{trans}(C_i) = \#\text{count}\{Y, r : \text{not}_r.A(X, Y)\} > (m - n)$ where $m = |N_f(\mathcal{K})|$. Then, there are $m - n$ number of pair $(a, b)$ such that $a \notin A^{T_B}$ or $(a, b) \notin r^{T_B}$. It coincides the alternate semantics defined in 4.3 consequently contradicts the assumption $\mathcal{I}_B \not\models C_i$.

- $C_i \subseteq n \cdot r.A$, then $\text{trans}(C_i) = \#\text{count}\{Y, r : r(X, Y), A(Y)\} > n$. Hence, there are less than or equal $n$ number of pair $(a, y)$ such that $(a, y) \in r^{T_B}$ and $y \in A^{T_B}$. It contradict the assumption $\mathcal{I}_B \not\models C_i$.

- All remaining cases can be treated analogously.

- $\alpha \in A$. Due to the direct translation for each assertion, obviously $A \subseteq B$. Hence, it contradicts the assumption $\mathcal{I}_B \not\models C_i$. Furthermore, $I_B$ is consistent because $M \not\models M^U(\mathcal{K})$. Consequently $\{\mathcal{A}(a), \neg\mathcal{A}(a)\} \not\subseteq B$ and $\{r(a, b), \neg r(a, b)\} \not\subseteq B$ for all concept names $A \in (\mathcal{K})$, role names $r \in (\mathcal{K})$ and individuals $a, b \in \Delta$.

\[\square\]

\textbf{Lemma 5.18.} For each $B \in \mathcal{B}_K$ and $\mathcal{I}_B \models K^X$, there exists $M \in AS(\Pi(\mathcal{K}^X, \Delta)) \setminus \{M^U(\Pi(\mathcal{K}, \Delta))\}$ such that $B = \beta_K(M)$.

\textbf{Proof.} From Proposition 5.13, we know there exists a model candidate $M \in AS(\Pi(\mathcal{K}^X, \Delta))$ where $B \subseteq M$. We show this $M$ does not imply the existence of atom $\text{incons}$, which causes $M \not\models M^U(\mathcal{K}, \Delta)$. The only possibilities to have an atom $\text{incons}$ in $AS(\Pi(\mathcal{K}^X, \Delta))$ is from axiom translation rules. Consider the rule $\Pi_\alpha$ is a translation for axiom $\alpha$. There are three possible cases:

- $\alpha \in R$. Then, $\alpha$ is of the form:
  - $\alpha = r \sqsubseteq s$, then $\Pi_\alpha = \text{incons} : - r(X, Y), \text{not}_s(X, Y)$. Since $\mathcal{I}_B \models \alpha$, then for any $a, b \in \Delta$, for each $r(a, b) \in B$, there is $s(a, b) \in B$. Hence, there is no grounded instance that will satisfy the rule.
  - $\alpha = \text{Dis}(r, s)$, then $\Pi_\alpha = \text{incons} : - r(X, Y), s(X, Y)$. Since $\mathcal{I}_B \models \alpha$, then for any $a, b \in \Delta$ there is no pair $(r(a, b), s(a, b)) \subseteq B$. Hence, there is no grounded instance will satisfy the rule.
  - $\alpha = r_1 \circ r_2 \sqsubseteq s$, then $\Pi_\alpha = \text{incons} : - r_1(X, Y), r_2(Y, Z), \text{not}_s(X, Z)$. Since $\mathcal{I}_B \models \alpha$, then for all $a_1, a_2, a_3 \in \Delta$, we have that if $r_1(a_1, a_2)$ and $r_2(a_2, a_3) \in B$, then $s(a_1, a_3) \in B$. Hence, there is no grounded instance will satisfy the rule.

- $\alpha \in T$. Then $\alpha$ is in the form $\top \sqsubseteq \bigcup_{i=1}^{m} C_i$. Since $\mathcal{I}_B \models \alpha$, then for any individual $a$, there is a $C_i$ with $1 \leq i \leq n$, such that $a \in C_i^{T_B}$. From the definition of $\Pi_{\text{chk}}(\mathcal{T})$, the rule is $\text{incons} : - \text{trans}(C_1), ..., \text{trans}(C_n), \text{thing}(X)$. We show $B$ does not activate $\text{incons}$. Let $C_i$ be the one that satisfies $a \in C_i^{T_B}$. Since $\alpha$ is normalized, then it is in the one of the form:
  - $C_i = A$, then $a \in A^{T_B}$ and $\text{trans}(C_i) = \text{not}_A(X)$. Consequently, $A(a) \in B$, hence $\text{not}_A(X)$ is not satisfied for any grounded-instance of $X = a$.
Lemma 5.19. For any $\Pi \in \Pi^\prime(K^X, \Delta)$, we have $|AS(\Pi^\prime(K^X, \Delta)) \setminus \{M^U(K, \Delta)\}| = |\{B \mid B \in B_K \text{ and } I_B \models K^X\}|$

Proof. From Lemma 5.13, we know that $\Pi_{gen}(K, \Delta)$ spans exactly all possibilities of $B \in B$ and $M^U(K, \Delta)$. However, we do not consider $M^U(K, \Delta)$ here. The question is whether there is a possibility that there exists more than one $M$ that satisfies $\beta_K(M) = B$. We know it is not possible since Proposition 5.15 holds.

We can conclude that $\Pi^\prime(K^X, \Delta)$ indeed computes the model of $K^X$.

Theorem 5.20. For any SROIQ knowledge base $K = (T, A, R)$, a fixed-domain $\Delta$, and $X \in 2^{U(\Pi(K, \Delta))}$,

$$AS(\Pi^\prime(K^X, \Delta)) \setminus \{M^U(K, \Delta)\} \cong \{B \mid B \in B_K \text{ and } I_B \models K^X\}$$

Proof. Follows immediately from Lemma 5.17, Lemma 5.18, and Lemma 5.19.
After showing what $\Pi''(K^X, \Delta)$ computes, we can go back to $\Pi'(K^X, \Delta)$. Now, we can see that $\Pi''(K^X, \Delta)$ has an answer set if $K^X$ is inconsistent.

**Lemma 5.21.** Let $K$ be a knowledge base, $\Delta$ be a fixed-domain, and $X \in 2^{U(\Pi(K, \Delta))}$. Then,

1. $\mathcal{AS}(\Pi'(K^X, \Delta)) = \{ MU(K, \Delta) \}$ if $K^X$ is inconsistent.
2. $\mathcal{AS}(\Pi'(K^X, \Delta)) = \emptyset$ if $K^X$ is consistent.

**Proof.** We prove each statement holds.

1. Assume $K^X$ is inconsistent. Then, there is no classical model $\Pi'(K^X, \Delta)$ that represents an interpretation. However, we know $MU(K, \Delta)$ is a classical model of $\Pi'(K^X, \Delta)$, hence a minimal one. Thus, $\mathcal{AS}(\Pi'(K^X, \Delta)) = \{ MU(K, \Delta) \}$.

2. Assume $K^X$ is consistent. Then, there exists constructible interpretations $B \in \mathcal{B}_K$ such that $I_B \models K^X$. Due to Lemma 5.18 then for each such $B$, there exists an $I \in \mathcal{AS}(\Pi''(K^X, \Delta))$ such that $B_K(I) = B$. Hence, we know that $MU(K, \Delta)$ is not a minimal model because obviously $I \not\subseteq MU(K, \Delta)$. Furthermore, there is no other possible answer set because of the correspondence. However, $\Pi_{inc}$ will eliminate all $I$ because $incons \not\in I$. It leaves us with no answer set for $\mathcal{AS}(\Pi'(K^X, \Delta))$. Thus, $\mathcal{AS}(\Pi'(K^X, \Delta)) = \emptyset$.

Finally, we can check back the complete encoding $\Pi(K, \Delta)$. We show $\Pi(K, \Delta)$ computes all inconsistent subsets of the knowledge base.

**Theorem 5.22.** $\mathcal{AS}(\Pi(K, \Delta)) = \{ X \in 2^{U(\Pi(K, \Delta))} \cup MU(K, \Delta) \mid K^X \text{ is inconsistent} \}$.

**Proof.** We show the proof with help of the splitting theorem. Let $U(\Pi(K, \Delta)) = \{ \text{activated}(i) \mid \text{activated}(i) \in \Pi(K, \Delta) \}$. We can build each solution $(X, Y)$ to $\Pi(K, \Delta)$ w.r.t. $U$. We know that $X \in 2^{U(\Pi(K, \Delta))}$ due to 5.5. Since 5.21 holds, there are two possibilities:

- $K^X$ is inconsistent. Then, $\mathcal{AS}(\Pi'(K^X, \Delta)) = \{ MU(K, \Delta) \}$. Thus, we have a solution $(X, MU(K, \Delta))$. Then, $X \in 2^{U(\Pi(K, \Delta))} \cup MU(K, \Delta)$ is a consistent answer set of $\Pi(K, \Delta)$.

- $K^X$ is consistent. Then, $\mathcal{AS}(\Pi'(K^X, \Delta)) = \emptyset$. Thus, we do not have any solution.

There is no other possible answer set $X$ for $b_U(\Pi(K, \Delta))$, hence no other possible solution.
6 Implementation and Evaluation

In this chapter, we discuss the implementation of all methods described earlier. Then, we provided several test cases to see how both approaches perform. Finally, we drew some conclusions from the result.

6.1 Implementation

We realized the glass-box method described in Chapter 4 in the Wolpertinger system as a class called `DebugTranslation`. In this way, we were able to reuse some parts of the naïve translation, such as OWLAPI integration and normalization. Naturally, we used Clingo as the ASP solver since Wolpertinger and asprin use Clingo. Wolpertinger itself already has a Clingo integration feature. Figure 6.1 depicts an overview of the system.

We used HermiT normalization with some adaption. At the moment, a user has to solve the ASP program manually with asprin.

![Figure 6.1: Overview of Wolpertinger](image)

The evaluation was done on a system with Intel i7-3630QM (2.4GHz, up to 3.4GHz) Processor, 8 GB of DDR3 RAM and Windows 10. The glass-box implementation used asprin 3.0.0 on the top of Clingo 5.2.0 with default settings for both tools. The asprin itself works on Python 2.7 and uses the Clingo module that pre-compiled and can be found in the package.

We implemented separately the black-box algorithm using the `explanation-workbench` 3.0.0 framework and used HermiT 1.3.8.413 as the reasoner for standard semantics.
Wolpertinger itself has a feature to axiomatize a knowledge base $\mathcal{K}$ with domain $\Delta$. Most of the cases we went with default settings where $\Delta$ is the set of all named individual in $\mathcal{K}$.

### 6.2 Evaluation Method

While there are many test cases that can be used to evaluate algorithms for finding justifications, usually most of them are designed under the standard semantics. We can get some of them from OWL 2 Test Case collection site[^1]. Usually they are designed to have an unsatisfiable class and give an anonymous instance of this class, hence the inconsistency. This is a problem because we can not handle anonymous individuals yet. However, we can give a concrete individual and fix the domain for testing them under our tool.

Other evaluation will use some problem encodings designed for fixed-domain semantics. We can simply axiomatize them to use standard semantics reasoners.

### 6.3 Unsatisfiable Role Chain Evaluation

In the fixed-domain reasoning encoding evaluation, an ontology that represents an unsatisfiable role chain is used [GRS16]. We reused the ontology as one of our evaluations. We started by defining $\mathcal{K}_n = (\mathcal{A}_n, \mathcal{T}_n, \emptyset)$, where:

$$\mathcal{T}_n = \{A_1 \sqsubseteq \exists r.A_2, \ldots, A_n \sqsubseteq \exists r.A_{n+1}\} \cup \{A_i \cap A_j \sqsubseteq \bot \mid 1 \leq i < j \leq n + 1\}$$

$$\mathcal{A}_n = \{A_1(a_1), \top(a_1), \ldots, \top(a_n)\}$$

$\mathcal{K}_n$ represents a $r$-chain from $\mathcal{A}_1$ to $\mathcal{A}_{n+1}$ and disjointness for each possible pair of concepts. Unsatisfiability is obtained by providing not enough individuals to fulfill the chain. Notice that under standard semantics, $\mathcal{T}_n$ is consistent.

The whole knowledge base (for any $n$) itself is a justification for the inconsistency, hence the only one. Although we know this information, we asked each tool to find all justifications, instead of one justification. The running time for both queries themselves does not differ so much. Intuitively, this happens because when an algorithm can stop immediately when it already know the whole knowledge base is a justification. However, for other cases it possibly differs much. Consider an example with one justification but not the whole knowledge base itself. After a tool have found the justifications, it still needs to make sure there is no other justification.

Table [6.1] shows the result for both black-box and glass-box approaches. The time did not increase so much from size 5 until 7, it seems running time are more defined by the overhead of each approach. We could see the glass-box approach consistently performed better for size 8 - 10. However, the pattern does not hold for size 11 where black-box

[^1]: http://owl.semanticweb.org/page/Category:InconsistencyTest.html
6.4 Insufficient Individual Evaluation

Table 6.1: Runtime for Unsatisfiable Role Chain.

<table>
<thead>
<tr>
<th># Instances</th>
<th>$\mathcal{K}_5$</th>
<th>$\mathcal{K}_6$</th>
<th>$\mathcal{K}_7$</th>
<th>$\mathcal{K}_8$</th>
<th>$\mathcal{K}_9$</th>
<th>$\mathcal{K}_{10}$</th>
<th>$\mathcal{K}_{11}$</th>
<th>$\mathcal{K}_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0.34s</td>
<td>2.18s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0.41s</td>
<td>2.38s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0.40s</td>
<td>2.70s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0.80s</td>
<td>4.40s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>7.48s</td>
<td>20.31s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>134.99s</td>
<td>184.511s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>3516.01s</td>
<td>2225.26s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

performance is much better than glass-box. TO (timeout) means the program could not solve the problem in 1 hour.

6.4 Insufficient Individual Evaluation

We designed a knowledge base with a controlled number of fixed-justifications. We exploited the cardinality restriction under fixed-domain semantics by not giving enough individuals. The idea is to create a root concept $C$ such that a member of this concept has to be connected to $n$ members of concept $A_i$. However, we only give $n$ free individuals in the domain and each concept is disjoint to the rest. Consequently, the existence of two axioms for $A_i$ and $A_j$, with $i \neq j$ leads to inconsistency.

We formalize this idea in a knowledge base $\mathcal{K}_{m,n} = (\mathcal{A}_n, \mathcal{T}_{(m,n)}, \emptyset)$ with domain $\Delta_n$, where:

$$
\mathcal{T}_{(m,n)} = \{ C \sqsubseteq \leq n \ r.A_1, ..., C \sqsubseteq \leq n \ r.A_m \} \cup \\
\{ C \sqsubseteq \neg A_1, ..., C \sqsubseteq \neg A_n \} \cup \\
\{ A_1 \sqsubseteq \neg A_2, A_1 \sqsubseteq \neg A_3; ..., A_{n-1} \sqsubseteq \neg A_n \}
$$

$$
\mathcal{A}_n = \{ C(a) \}
$$

$$
\Delta_n = \{ a, x_1, ..., x_n \}
$$

Any $\mathcal{K}_{m,n}$ has $\binom{m}{2}$ $\Delta_n$-justifications for inconsistency. We obtain

$$
\mathcal{J}^* = \{ C(a), C \sqsubseteq \leq n \ r.A_i, C \sqsubseteq \leq n \ r.A_j \mid i < j < n \}
$$

Thus, $\mathcal{J}^*$ is the set of all $\Delta$-justifications for inconsistency of $\mathcal{K}_{m,n}$. Furthermore, the number of free individuals $n$ does not contribute to the size of $\mathcal{J}^*$. Obviously, greater $n$ resulted in longer computation time. The knowledge base itself is consistent under standard semantics. The number of justifications can easily be raised by manipulating combination coefficient. For example, if we have $n + 1$ individuals, we will have $\binom{n+1}{3}$ justifications.
We created some instances for this problem and tested them with available tools by asking for all inconsistency justifications. We checked for how both the number of concepts and individuals contribute to computation time. The result is shown in Table 6.2. A timeout (TO) means the program could not solve the problem in 1 hour. In the table, one stands for asking one justification and all for asking all justifications.

In this evaluation, we could see the glass-box approach worked very well and black-box did not. We also noticed that increasing the number of individuals has really big impact on the black-box approach. We guess that happens because HermiT is not optimized for cardinality restrictions under a domain with bounded size. It worked fine only for test cases less than six individuals.

### 6.5 Inconsistent Sudoku

We reused Sudoku encoding knowledge base that was used in previous work [GRS16]. However, we deliberately filled some cells incorrectly. For example, putting two numbers of 8 in the same row or two numbers in the same cell. Looking for justifications is not an easy but feasible task. However, it is very difficult for the tools to make sure there is no more justification left. For that reason, we set a timeout and check how many justifications we could get. We gave 30 minutes timeout for both approaches.

Empty board and 46-filled cells instance are taken as the base of the evaluation. We added several misplaced numbers to the base boards. While we obviously know the number that we added, the number of justifications for inconsistency is not trivial. For example, if we put number 8 in cell (1,2) when there is already number 8 in cell (1,1), they violated two rules for row and square. *sudoku-empty* stands for an empty board before we put misplaced numbers, while *sudoku-filled* stands for a full configuration of sudoku with one solution.

The results are varying where adding more misplaced numbers did not always give more justifications. It is also has to be noted that the interval between finding each justification are not the same. In the black-box approach, the first justification was
found within 10 seconds, but the third justification was found at around 5 minutes mark. Meanwhile, both the first and third justification were found around 5 minutes mark by glass-box approach.

6.6 Analyzing the Result

It is very hard to see a clear conclusion from the result, as each approach has cases where it performs better. Despite the black-box approach failed to get any result in Section 6.4, it did fine in other evaluations. Hence, it is feasible to find justifications using existing frameworks such as Protégé. However, there is a lack of control that can be configured by the users using this approach.

Considering the evaluation in Section 6.4, it is quite unexpected that there are big differences in the result between two approaches. If we look closer, the knowledge base consists of cardinality axioms that have unique behavior in fixed-domain semantics. We said unique because quantification axioms is delimited by the size of the domain. In standard semantics, an algorithm will simply introduce a new individual if there are not enough individual can fulfill the condition. However, it is not possible to do so in the fixed-domain semantics. We guess HermiT does not take care of this case. Meanwhile, ASP solver can compute this better since cardinality aggregate function is native in Clingo.

On the other hand, there is a possibility to optimize the ASP translation approach. For example, Clingo has a parallelization feature which is easy to use. In the evaluation of this work, we only explored the default configuration of Clingo. The tool itself provides quite many settings. It is possible to tune Clingo even more to increase performance. One thing to consider using this approach is the memory usage. We found that ASP approach used extensive memory.

The result in Section 6.5 convinced us it is very hard to compute all justifications. Although, it is feasible to compute one or several justifications. While it restrained us to do a full repair of the knowledge base, we can do an incremental repair. It is also possible that some justifications will go away by fixing a justification. Furthermore, giving a right simple repair is very hard because the machine cannot understand the real world semantics that represented by each axiom.
7 Conclusion and Future Work

7.1 Conclusion

Modeling is a process that is prone to error. There exists an enormous number of studies to assist such a process in DLs. While there are several notions to describe a set that is responsible for an entailment, we used the justification-based explanation framework. On the other hand, we have the fixed-domain semantics for description logics which has better complexity for expressive DLs. The models are arguably more intuitive in some ways when modeling with DLs.

In this thesis, we looked deeper into the intersection of two previous topics. We defined and analyzed justification framework under fixed-domain semantics, which we call Δ-justification for a domain Δ. Furthermore, we focused more on the justifications of inconsistency in a knowledge base. Since we have more restricted models, there are possibly more responsible justifications. We showed that an inconsistency justification in standard semantics is a justification under fixed-domain semantics. It is possible to use existing justification tools using axiomatized knowledge base, such as the out-of-the-box feature in Protégé. We recognize this method as the black-box approach.

While black-box method is easy to use, we looked for other possibilities to exploit the ASP translation for fixed-domain reasoning. We designed a glass-box approach with translating a knowledge base to an ASP program such that each answer set corresponds to an inconsistency justification. Furthermore, the user can choose whether they want to check the taxonomy (TBox and RBox) or the data (ABox). We utilized asprin, a tool for getting preferred answer sets, when designing the encoding.

We showed the soundness and correctness of the program, i.e., each answer set has a corresponding justification and vice versa. We employed the splitting theorem to simplify the proof into two parts. The first part guesses the subset of the knowledge base, while the second part checks the satisfiability of such a subset. We made use of the unified model, which represents the principle of explosion in the logical system. It is definitely a classical model of the program, but only considered as a stable model if there is no other model, i.e., the knowledge base is inconsistent.

As mentioned previously, we can employ black-box approach in Protégé. However, for the sake of better evaluation, we used explanation-workbench combined with axiomatization tool. In previous work, the ASP encoding for fixed-domain semantics is realized in a tool called Wolpertinger. Thus, we embedded glass-box encoding in Wolpertinger as a debugging feature. Unlike the other features that only need Clingo, users need asprin to use it. Several configurations are available to change which part of the knowledge base is checked.

1https://github.com/wolpertinger-reasoner/Wolpertinger
base that we want to check: TBox + RBox, ABox, or all of them.

We evaluated both approaches for several test cases. There is no dominating approach in general, but we found a case which is not feasible for black-box approach, but can be done by glass-box. Furthermore, there is still a big space for improvement and optimization in glass-box. For example, we can impose parallelization feature which is native for Clingo.

7.2 Further Work

7.2.1 Optimizing Existing Approach

There are many possibilities to optimize each approach, especially for the glass-box approach. As mentioned previously, there are many unexplored settings that can be used to accelerate the computation. One promising idea is the parallelization feature in Clingo. From the nature of the problem, computing ASP programs gets a big acceleration from parallelization. It can make bigger problems to be more feasible using this approach. From the translation side itself, it is possible to add some “helper” rules. For example, adding \( C(X) :- D(X) \) for simple subsumption. Currently we do not know whether Clingo already can infer such case or not.

7.2.2 User Friendly Justification Tool

So far, any approach does not give a huge control for the user. For a big knowledge base, this kind of flexibility is very important when it is not possible to get all justifications. For example, if the user knows a part of a knowledge base is fine, they might want to exclude them and shorten the computation time. The feedback of finding justifications for glass-box approach is not yet user-friendly either. The tool gives the number of the axioms that coincide with the a justification. Thus, a user has to cross-check what are the original axioms for those rules. It is not a trivial task to implement because of we reuse existing normalization code where such linking is not needed. Furthermore, an interactive tool is really nice to provide.

7.2.3 Finding Fixed-Justifications for Arbitrary Axioms

In this work, we consider only fixed-justifications for knowledge base inconsistencies. We believe that this is a good starting point for finding fixed-justifications for any axiom. It is known that one can reduce an entailment into knowledge base satisfiability [HPS03]. Consider a simple subsumption example \( C \sqsubseteq D \). We can insert an assertion \( (C \sqcap \neg D)(a) \) for a new individual \( a \) and ask for the satisfiability. However, we need to investigate is it possible to do this in fixed-domain reasoning.


Bibliography


