Constraint-based Reasoning for Description Logics with Concrete Domains and Aggregations

by

Wudhichart Sawangphol

Thesis

Submitted by Wudhichart Sawangphol

for fulfillment of the Requirements for the Degree of

Doctor of Philosophy (0190)

Supervisor: Dr. Yuan-Fang Li

Associate Supervisor: Dr. Guido Tack

Faculty of Information Technology
Monash University

August, 2017

Contents

List of Tables

List of Figures

Constraint-based Reasoning for Description Logics with Concrete Domains and Aggregations

Wudhichart Sawangphol
[redacted]
Monash University, 2017

Supervisor: Dr. Yuan-Fang Li
[redacted]
Associate Supervisor: Dr. Guido Tack
[redacted]

Abstract

Expressive Description Logics (DLs) are used to describe, in the form of ontologies, many real-world phenomena such as biology and biomedicine.

Automated DL reasoning is used to maintain quality of an ontology and deduce implicit knowledge encoded in DL based ontologies. Tableau calculi are the mainstream reasoning algorithms for expressive DLs. However, the efficiency of tableau-based algorithms is still a bottleneck for large and complex ontologies. Another limitation of these algorithms is that any extension of the language typically requires significant re-engineering effort.

This thesis makes two main contributions to the field of Description Logic. Firstly, we propose a novel decidable Description Logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ that supports concrete domains and aggregations. Concrete domains and aggregations allow us to model complex knowledge that involves concrete values such as numbers and their aggregations (min, max, count, and sum). Reasoning over DLs with concrete domains and aggregations has not received much attention, due to the inherent complexity of certain reasoning tasks as well as difficulties in extending existing DL reasoning algorithms. We show that some fundamental reasoning tasks of the new Description Logic are NP-COMPLETE.

Secondly, we propose a new DL reasoning framework based on Constraint Programming (CP) techniques. This CP-based framework supports our new Description Logic as well as the most well-known Description Logic $\mathcal{ALC}$. We show that our approach is sound and complete.

The long-standing research in CP has made tremendous progress. CP has been developed to handle numerical constraints and offers powerful search and solving techniques that can reduce the search space effectively.

Our reasoning framework encodes Description Logics to concise and straightforward MiniZinc, a standard modelling language for modelling CP problems. Thus, we are able to employ mature CP solving and optimisation techniques to improve reasoning efficiency for DLs with and without aggregations. To the best of our knowledge, our proposed approach is the first implementation that provides reasoning support for DLs with concrete domains and aggregations. Our comprehensive evaluation also shows that our approach is competitive against, and sometimes superior to, state-of-the-art DL reasoners for some ontologies. Furthermore, our approach is feasible and effective for our new Description Logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$.

Constraint-based Reasoning for Description Logics with Concrete Domains and Aggregations

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Wudhichart Sawangphol
August 14, 2017

Acknowledgments

I owe thanks to many people who have supported me during my PhD. Firstly, I would like to thank my supervisors, Dr. Yuan-Fang Li and Dr. Guido Tack, who made corrections and helped bring this research to a completion. Without the useful comments and crucial support of them, this project could not be finished. They, always offered me time, suggestions, and constant encouragement during the time I conducted this research.

I am also grateful to Faculty of Information and Communication Technology, Mahidol University, Thailand for their funding and supports during my PhD study.

I would like to thank my beloved Sawangphol family especially my parents, Benjapen and Natachok Sawangphol for their supports and encouragement which mean so much to me. In addition, I would also like to extend my thankfulness to Ketmany Vilayvong, my girlfriend for her support.

I would like to thank my colleagues at Monash University whom I shared interesting discussions and wondelful time in Australia. I wish to mention in no particular order Senthooran, Omid, Tennyson, Tian, Ariesta, Ranjie, Yuri, Prajwol, and Kevin.

Moreover, I would like to extend my thankfulness to my Thai friends who always give me helps and suggestions when I need. I also wish to mention them here Boonlom, Narumon, Chantanee, Kanlaya, Sirakul, Wanchana, and Thanabhorn.

Finally, I would like to thank the staffs of Office of Educational Affairs, Royal Thai Embassy, Australia for their supports and helps.

Wudhichart Sawangphol

Monash University
August 2017

Chapter 1

Introduction

Description Logics (DLs) are a family of logic-based knowledge representation formalisms which aim at representing knowledge of application domains by defining concepts of the domain, relationships of the concepts and instances in the concepts. There have been significant research in this area in two main directions. The first direction is to extend Description Logics with some language constructs in order to improve their expressiveness. The second direction is to investigate algorithms fro efficient reasoning about Description Logics.

This thesis investigates both directions. First, we have developed a novel Description Logic that allows numeric constraints and aggregations be placed on concepts. Second, we have explored a reasoning paradigm by translating Description Logics to Constraint Programming (CP) constraint models, which can be efficiently solved by CP solving techniques.

1.1 Description Logics with concrete domains and aggregations

Description Logics (DLs) are a family of knowledge representation languages that provide semantics to model relationships between objects in particular domains in a logical manner (Baader & Nutt, 2003). In the *Semantic Web*, DLs provide the theoretical foundation for standard ontology languages, OWL 1 (Horrocks, Patel-Schneider, & van Harmelen, 2003) and OWL 2 (Grau et al., 2008), and serve as the basis of the development of ontology reasoning algorithms. DLs have became more important due to the advent of Semantics Web. In addition, DLs have been successfully exploited in various application domains such as biology (Ashburner et al., 2000; Sidhu, Dillon, Chang, & Sidhu, 2005; Osumi-Sutherland et al., 2012), medicine (Stearns, Price, Spackman, & Wang, 2001; McBride, Lawley, Leroux, & Gibson, 2012), and pervasive computing (Wang, Zhang, Gu, & Pung, 2004; Chen, Perich, Finin, & Joshi, 2004).

A Description Logic defines *axioms* that state how certain *concepts* are related to each other, where concepts can be considered sets of abstract objects called *individuals*. Concepts are typically defined using constructs such as conjunction and disjunction, as well as quantification over binary relations, so-called *roles*.

A DL knowledge base (or ontology) generally consists of a set of statements, or *axioms*. Axioms are used to capture partial knowledge of a particular domain. There are three main groups of axioms. The first one is *Terminological* axioms (TBox), which is the schema of the knowledge base. The second one is *Assertional* axioms (ABox), which is an instantiation of the terminological schema. The third one is *Role characteristics* (RBox), which expresses axioms about roles. The complexity of such a knowledge base depends on not only the language constructs, but also the combination of these three types of axioms and how they are defined.

Description Logics have many different language constructs for describing concepts of domains and relations among them. Each different combination of language constructs may yield a new Description Logic. The most well-known Description Logic is $\mathcal{ALC}$, which is a notational variant of the Modal Logic K_m (Schmidt-Schauß & Smolka, 1991).

For many years, the complexity and properties of different combination of language constructs have been studied. Researchers have tried to balance between the expressive power and complexity of Description Logics. More expressive extensions of $\mathcal{ALC}$ have been proposed, mostly in the abstract domain. Some well-known Description Logics are $\mathcal{ALCQ}$ (Hollunder & Baader, 1991), $\mathcal{SHOIQ}$ (Horrocks & Sattler, 2005), and $\mathcal{SROIQ}(\mathcal{D})$ (Horrocks, Kutz, & Sattler, 2006). These extensions improve the expressive power of Description Logics in order to support real-world applications. For example, $\mathcal{SROIQ}(\mathcal{D})$ extends $\mathcal{ALC}$ with qualified number restriction, role characteristics such as transitive role and role hierarchy, nominals, and data types. It is the base Description Logic of the OWL 2 ontology language (Grau et al., 2008).

Restrictions on $\mathcal{ALC}$ have also been investigated, resulting in less expressive but tractable DLs. For instance, the $\mathcal{EL}$-family ($\mathcal{EL}$, $\mathcal{EL}^+$, and $\mathcal{EL}^{++}$) of logics (Baader, 2003; Baader, Brandt, & Lutz, 2005; Baader, Lutz, & Brandt, 2008) are tractable, but expressive enough to model several important ontologies such as the Gene Ontology (GO) (Ashburner et al., 2000), the Systematized Nomenclature of Medizine, Clinical Terms ontolgy (SNOMED-CT) (Stearns et al., 2001), and the thesaurus of the National Cancer Institute (NCI) ontology (Golbeck et al., 2003). Example 1.1.1 shows a simple DL ontology in $\mathcal{ALC}$.

Example 1.1.1. An example ontology about pizzas $\mathcal{O}_{pizza}$, inspired by the Pizza Ontology (Drummond, Horridge, Stevens, Wroe, & Sampaio, 2007), is defined as follows:

$$\begin{aligned}
\mathsf{VegetableTopping} &\sqsubseteq \neg\mathsf{MeatTopping} && \text{(A1)}\\
\mathsf{VegetableTopping} &\sqsubseteq \neg\mathsf{FishTopping} && \text{(A2)}\\
\mathsf{MeatFishPizza} &\equiv \mathsf{Pizza} \sqcap \\
&\quad \exists\mathsf{hasTopping}.(\mathsf{MeatTopping} \sqcup \mathsf{FishTopping}) && \text{(A3)}\\
\mathsf{VegetarianPizza} &\equiv \mathsf{Pizza} \sqcap\ \forall\mathsf{hasTopping}.\mathsf{VegetableTopping} && \text{(A4)}
\end{aligned}$$

This ontology introduces the concepts VegetableTopping, MeatTopping, FishTopping, Pizza, MeatFishPizza, and VegetarianPizza, and defines axioms that constrain their sub-concept (subsumption) and equivalence relationships. For example, a VegetableTopping is subsumed by a negated MeatTopping, which simply means that all individuals that are classified as VegetableTopping are not MeatTopping. The symbol $\sqsubseteq$ denotes a sub-concept (subsumption) relationship and a negation is denoted as $\neg$. The concept MeatFishPizza is defined as those individuals that are classified as Pizza, and for which there exists a successor individual in the hasTopping binary relation (role) that is classified as either MeatTopping or FishTopping. The concept VegetarianPizza is defined as those individuals that are classified as Pizza, and for which all successors in the hasTopping relation (role) that are classified as VegetableTopping. The symbol $\equiv$ denotes an equivalence relationship, the symbol $\exists$ represents an existential quantifier, the symbol $\forall$ represents a universal quantifier, an intersection (conjunction) is denoted as $\sqcap$, and a union (disjunction) is denoted as $\sqcup$. □

Another important extension is *concrete domains* (Lutz, Areces, Horrocks, & Sattler, 2003), such as number and string, and *aggregations* (Baader & Sattler, 2003) such as sum and count. Aggregation over concrete domains is a very natural extension to Description Logics, as it allows concepts and individuals to be organised by their physical attribute values. Examples of DL that is extended by these extensions are $\mathcal{ALC}(\mathcal{D})$ (Baader & Hanschke, 1991) and $\mathcal{ALC}(\Sigma)$ (Baader & Sattler, 2003). These extension helps to express some phenomena more precisely and directly. However, extensions on *concrete domains* and *aggregations* have not received much attention. Currently, DLs have been mostly developed over the abstract domain due to the difficulty in extending the dedicated tableau reasoners. Lack of such DLs motivated our research. It is interesting to revisit and investigate aggregations as concept constructs further since the ability to express aggregation over concrete domains is useful in many domains and situations.

The Description Logic with concrete domains and aggregations $\mathcal{ALC}(\Sigma)$ was first proposed in (Baader & Sattler, 2003). This Description Logic is undecidable. In order to retain decidability, syntactic restrictions were placed on $\mathcal{ALC}(\Sigma)$, resulting in the decidable Description Logic $\mathcal{EL}(\Sigma)$ (Baader & Sattler, 2003). However, this logic has been designed to express only concept descriptions, and it does not allow us to express knowledge bases (or ontologies) involving concrete domains and aggregations. If we can develop a Description Logic with concrete domains and aggregations to express knowledge bases, the knowledge bases will be more precise and useful. Let us use Example 1.1.2 to illustrate the usefulness of concrete domains, in this case natural numbers and their aggregations. Note that other examples with more complex concrete domain and aggregations will be presented in Chapter 6.

Example 1.1.2. Suppose we want to model knowledge about calories of pizzas. For example, we can add the following axioms to $\mathcal{O}_{pizza}$ in Example 1.1.1:

$$\begin{aligned}
\mathsf{Pizza} &\sqsubseteq\ =.(\mathsf{calories}, \mathsf{sum}(\mathsf{hasTopping} \circ \mathsf{calories})) && \text{(A5)}\\
\mathsf{HealthyPizza} &\sqsubseteq \mathsf{Pizza} \sqcap \leq_{400} .(\mathsf{calories}) && \text{(A6)}\\
\mathsf{MeatTopping} &\equiv\ \geq_{200} .(\mathsf{calories}) \sqcap \leq_{400} .(\mathsf{calories}) && \text{(A7)}\\
\mathsf{FishTopping} &\equiv\ \geq_{50} .(\mathsf{calories}) \sqcap \leq_{100} .(\mathsf{calories}) && \text{(A8)}
\end{aligned}$$

With aggregations, we then can define the concept Pizza as those individuals that have calories that are equal to the sum of calories-values of hasTopping-successors, in A5, assuming calories is a concrete domain feature. With concrete domains, the concept HealthyPizza is defined as those individuals that are classified as Pizza and have calories less than 200, in A6. The concept MeatTopping is defined as those individuals that have calories between 100 and 200, in A7, and the concept FishTopping is defined as those individuals that have calories between 50 and 100, in A8. □

Goal 1. Our first goal is to develop a novel decidable Description Logic that supports concrete domains and aggregations for modelling knowledge bases (or ontologies). Our work investigated the combination of different abstract syntax, concrete domains, aggregations, and types of TBox and proposed a novel decidable DL that supports concrete domains and aggregations $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. In addition, we explored the reasoning services that would arise from this Description Logic.

1.2 Reasoning in Description Logics

Automated reasoning, a major research topic in DL, is used to maintain quality of an ontology and deduce implicit knowledge encoded in an ontology. It also helps debug an ontology.

The consistency checking task determines whether there is no logical contradiction in an ontology, while the concept satisfiability task checks whether a particular atomic concept in the ontology is empty or not, i.e., if a concept can contain instances. The concept subsumption task checks whether one concept subsumes another concept, whereas the classification task identifies subsumption relationship between all pairs of atomic concepts. For highly expressive Description Logics, the complexity of core reasoning tasks (also known as inference problems) which include *concept satisfiability*, *concept subsumption*, *classification* and *consistency checking* (Baader & Nutt, 2003), is very high (NP-complete, PSpace-complete, ExpTime-complete and even more) (Donini, 2003; Kazakov, 2008). Example 1.2.1 illustrates the usefulness of DL reasoning.

Example 1.2.1. Extending the ontology $\mathcal{O}_{pizza}$ in Examples 1.1.1 and 1.1.2, ontology engineers want to define the concept MixedToppingPizza, which should be a sub-concept of a pizza that has some meat, fish, and vegetables as toppings in $\mathcal{O}_{pizza}$. However, they define MixedToppingPizza in axiom A9, which is incorrect.

$$\mathsf{MixedToppingPizza} \sqsubseteq \mathsf{VegetarianPizza} \sqcap \mathsf{MeatFishPizza} \tag{A9}$$

From axioms A1 and A2, it shows that VegetableTopping and MeatTopping are disjoint and VegetableTopping and FishTopping are disjoint. Thus, ontology reasoning can infer that VegetarianPizza and MeatFishPizza are disjoint in $\mathcal{O}_{pizza}$ since MeatFishPizza can have either MeatTopping or FishTopping as toppings but VegetarianPizza can have only VegetableTopping as toppings. Due to this inference, ontology reasoning can discover an error that MixedToppingPizza, which is subsumed by both VegetarianPizza and MeatFishPizza, is *unsatisfiable*. Axiom A9 should be modified as follows to correctly model MixedToppingPizza.

$$\mathsf{MixedToppingPizza} \sqsubseteq \exists \mathsf{hasTopping.VegetableTopping} \sqcap \mathsf{MeatFishPizza} \tag{A10}$$

In addition, with concrete domains and aggregations (A5-A8), we can ask whether MeatFishPizza is HealthyPizza (i.e., MeatFishPizza $\sqsubseteq$ HealthyPizza, which is true. □

Significant research has been devoted to investigating algorithms to effectively reason over Description Logics. DL reasoning can be viewed as an AND-OR search problem on directed graphs (Donini, 2003). The *tableau-based* algorithms (Schmidt-Schauß & Smolka, 1991) are the main approach employed with many optimisation techniques for expressive Description Logics (Horrocks, 2003; Motik, Shearer, & Horrocks, 2007b, 2007a, 2009). Such algorithms attempt to construct a model of an ontology as a search graph where a branch is terminated by logical contradictions. These algorithms have been implemented in many state-of-the-art DL reasoners such as FaCT++ (Tsarkov & Horrocks, 2006), Pellet (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007), and Konclude (Steigmiller, Liebig, & Glimm, 2014). HermiT (Shearer, Motik, & Horrocks, 2008; Glimm, Horrocks, Motik, Stoilos, & Wang, 2014) uses the hypertableaux calculus (Motik et al., 2007b, 2007a, 2009). However, the efficiency of tableau-based algorithms is still a bottleneck for large and complex ontologies. The major sources of this inefficiency are:

- OR-search (OR-branching) (Donini, 2003) is due to the presence of disjunctions. It may cause the evaluation of an exponential number of different candidate models when we want to find a model for the ontology.

- AND-search (AND-branching) (Donini, 2003) is due to the presence of existential restrictions. The exponential behaviour occurs because of the combination of existential and universal restrictions. It may result in a candidate model exponential in the size of an ontology when we want to find a clash.

- Complex role constructs, such as role inclusion, that may cause exponential blowup in the size of RBox (Kazakov, 2008).

- The combination of nominals and other language constructs such as qualified number restrictions and unqualified number restrictions (Motik et al., 2009; Zuo & Haarslev, 2013).

A further limitation of these algorithms is that any extension of the language typically requires significant re-engineering of the reasoning algorithm. Due to these limitations, only a few implementations exist for extensions such as concrete numerical domains. The tableau-based algorithm was proposed for DLs with concrete domain and aggregations (Baader & Sattler, 2003). However, no reasoning support had been implemented for aggregation.

Efficient *completion-based* algorithms have been developed for less expressive Description Logics such as the $\mathcal{EL}$ family (Brandt, 2004; Baader, Lutz, & Suntisrivaraporn, 2005). These algorithms have been implemented in many state-of-the-art DL reasoners such as CEL (Baader, Lutz, & Suntisrivaraporn, 2006) and ELK (Kazakov, Krötzsch, & Simancik, 2014). These algorithms are also extended to support $\mathcal{EL}$ with concrete domains (Baader, Brandt, & Lutz, 2005). Such algorithms have also been used for the inference problem of expressive DLs through syntactic approximation (Thomas, Pan, & Ren, 2010), which is incomplete. Consequently, reasoning on large and complex ontologies is still a computationally challenging problem and efficient reasoning procedures for expressive Description Logics are still required (Kang, Li, & Krishnaswamy, 2012; Samwald, 2013; Kang, Pan, Krishnaswamy, Sawangphol, & Li, 2014; Kang, Krishnaswamy, & Li, 2015).

Constraint programming (CP) is a powerful approach for solving combinatorial problems and it offers efficient techniques in a wide range of domains and applications such as scheduling, planning, vehicle routing, computer networks, and bioinformatics. (Rossi, van Beek, & Walsh, 2006). Further application areas include data mining (Guns, Dries, Tack, Nijssen, & De Raedt, 2013; Métivier, Loudni, & Charnois, 2013), robotics (Jaulin, 2016), and sensor networks (Bijarbooneh, Pathak, Pearson, Issarny, & Jonsson, 2014). CP supports many forms of constraints such as numerical constraints, Boolean constraints, and linear constraints.

A combinatorial problem consists of a set of decision variables and a set of constraints that model relations among the decision variables. Such a combinatorial problem is called *Constraint Satisfaction Problem* (CSP). The goal of constraint programming is to find a solution that satisfies all constraints by assigning a value to each variable. In general, the complexity of solving the constraint satisfaction problem is NP-COMPLETE (Feder & Hell, 2006).

In addition, in the last twenty years, we have witnessed an impressive improvement of the efficiency of CP-based techniques, which have been implemented in *constraint solvers* (CP solvers). There are many freely available CP solvers such as G12 solvers[1] (Stuckey et

[1]`http://www.minizinc.org/software.html`

al., 2005), Gecode[2] (Tack, 2009), Opturion CPX[3], and Chuffed[4] (Chu, 2011). Most modern constraint solvers implement a well-known constraint solving technique called *Constraint Propagation* (Fruhwirth & Abdennadher, 2006). The main idea of constraint propagation is to reduce the size of a problem by reducing the size of domains of the variables until all variables have only one value (solution), which satisfies all constraints (Bessiere, 2006; Fruhwirth & Abdennadher, 2006). However, constraint propagation is not complete. Therefore, it has to be interleaved with *search algorithms* to achieve completeness.

The needs from real-world applications have motivated significant research on solving techniques and constraint modelling languages. An advanced CP solving approach, *Lazy Clause Generation* (LCG), has been proposed (Ohrimenko, Stuckey, & Codish, 2007; Feydy & Stuckey, 2009). LCG combines the advantages of SAT solving such as efficient nogoods learning and backjumping with the advantages of CP solving such as efficient constraint propagation, and simple and powerful modelling. LCG has been implemented in CP solvers such as Opturion CPX and Chuffed. In addition, CP offers *search heuristic* techniques, which give an order of decision variables to be assigned, in order to improve the solving performance.

Additionally, Constraint Programming offers a simple but powerful modelling language, MiniZinc (Nethercote et al., 2007). MiniZinc is a medium-level declarative modelling language. In addition, MiniZinc has been developed as a standard modelling language for Constraint Programming problems. It is also high-level and expressive enough to express most CP problems.

As can be seen, the long-standing research in Constraint Programming (CP) (Rossi, Beek, & Walsh, 2006) has made tremendous progress. Constraint Programming (CP), which has been developed to handle numerical constraints and offer powerful search and reasoning techniques that can reduce the search space efficiently. In Description Logic, the concept constructs of concrete domain and aggregations can be considered as numerical constraints, where concrete domain is number. Therefore, we believe we can leverage CP-based techniques to tackle some sources of inefficiency of Description Logic reasoning, especially OR-search, and create reasoning support for Description Logics with concrete domains and aggregations. For example, a disjunction (cause of OR-search) can be encoded into a set union constraint. The underlying constraint solver decomposes the set union constraint into disjunctions of Boolean variables. The clause learning in LCG solvers such as Chuffed (Chu, 2011) can handle these disjunctions very efficiently.

Goal 2. Our second goal is to develop new techniques for reasoning in Description Logics, especially to support reasoning in concrete domain and aggregations. We achieve this through encoding Description Logic inference problems as Constraint Programming problems. We aim at exploiting the capabilities of modern CP solving techniques as an alternative to the traditional tableau-based algorithms. With this approach, advances of CP solving techniques will be freely available for reasoning of Description Logics. Our work investigated the efficiency and feasibility of encoding different reasoning services in Description Logics with and without concrete domain and aggregations using a CP-based approach. We defined a formal encoding for $\mathcal{ALC}$ and $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ with acyclic TBoxes and proved its soundness and completeness. We conducted evaluation of our CP-based reasoning framework on benchmarks and compare the performance with state-of-the-art reasoners for DL reasoning. Particularly, we started from the most well-known Description Logic $\mathcal{ALC}$, and then we moved to the Description Logic with concrete domain and

[2]http://www.gecode.org/
[3]http://www.opturion.com/cpx
[4]https://github.com/geoffchu/chuffed

aggregations. We also demonstrates the effectiveness of some CP optimisation techniques in improving reasoning efficiency.

1.3 Contributions

In this thesis, we propose a novel Description Logic with concrete domain and aggregations, and new reasoning techniques for Description Logics. In more detail:

1. We have developed a novel description logic namely, $\mathcal{ELU}^{(\neg)}(f,\Sigma)$, which includes functional features and aggregations on concrete domain (the concrete domain we consider is natural numbers). In addition, we have shown that our Description Logic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is decidable. This is achieved through some reasonable syntactic restrictions on $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. We also have shown that the complexity of reasoning tasks, concept satisfiability, consistency checking, and concept subsumption, of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ are NP-COMPLETE. This contribution addresses **Goal 1**.

2. We have developed an alternative approach to ontology reasoning by exploiting modern CP techniques. Ontologies are encoded into MiniZinc models in a direct and succinct way. Existing CP solvers can then be readily applied to perform ontology reasoning without modification. We illustrate our CP-based reasoning approach through the encoding of $\mathcal{ALC}$ and its sub logic. Our empirical evaluation on ontologies in the expressive DL $\mathcal{ALC}$ shows that our approach is competitive against existing ontology reasoners, outperforming some state-of-the-art tableau-based reasoners, sometimes by several orders of magnitude.

3. We have developed CP-based approach, which is the first and only implementation for any Description Logic with aggregations over concrete domains. In this thesis, $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is used as a sample of Description Logic with such features. Moreover, we have shown that our encoding approach is sound and complete.We also demonstrate that, with some simple model-level optimisation, our approach is feasible for supporting concrete domain and aggregation reasoning in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$.

The second and the third contributions address **Goal 2**.

1.4 Thesis Outline

The thesis consists of eight chapters including this one. The thesis is divided in two main parts. The first part, consisting of Chapters 2, 3 and 4, gives the necessary background and the state-of-the-art in Description Logics, reasoning algorithms, and Constraint Programming for the rest of the thesis. The second part, consisting of Chapters 5, 6 and 7, describes the theoretical, technical and practical results achieved of the new Description Logic and the encoding for Description Logics with and without concrete domain and aggregations, as the original contributions of this thesis. The rest of the chapters are structured as follows:

- Chapter 2 provides the necessary background information of Description Logics. In detail, we define notation that is used in this thesis. Then we provide the common syntax and semantics of Descriptions Logic together with some Description Logic families. In addition, we present the main reasoning services.

- Chapter 3 provides an overview of the state-of-the-art reasoning algorithms. In particular, we present the main notions of the tableau-based algorithms for Description Logics with and without concrete domain and aggregations. In addition, we present the main notions of the completion-based algorithms.

- Chapter 4 provides the main notions of Constraint Programming. In particular, we present Constraint Propagation, the modelling language MiniZinc, and some advanced solving and modelling techniques such as Lazy Clause Generation.
- In Chapter 5, we present a novel set-encoding in order to tackle the problem of consistency checking, concept satisfiability, and concept subsumption in $\mathcal{ALC}$ with respect to acyclic TBoxes. In addition, we present the soundness and completeness proofs of our encoding and the evaluation comparing with the state-of-the-art DL reasoners.
- In Chapter 6, we present a novel Description Logic with concrete domain and aggregations $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ together with examples of applications. In addition, we present inference problems that arise from this Description Logic. The decidability result is also presented.
- In Chapter 7, we present a novel set-encoding in order to tackle the problem of consistency checking, concept satisfiability, and limited concept subsumption in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ with respect to acyclic TBoxes. In addition, we present the soundness and completeness proofs of our encoding and some optimisation to improve the performance of reasoning together with the evaluations.
- The last chapter concludes the overall contribution of this thesis, and suggests some future research directions.

1.5 Publications

This section reports accepted and submitted papers arising from this thesis. Our paper accepted in the 15th International Workshop on Constraint Modelling and Reformulation (ModRef 2016) presents the CP-based reasoning framework.

- Sawangphol W., Li Y.-F., & Tack G. (2016), CP4DL: Constraint-based Reasoning for Expressive Description Logic. In *the 15th International Workshop on Constraint Modelling and Reformulation (ModRef 2016)* Toulouse, France.

In addition, the following paper submitted to the 26th International Joint Conference on Artificial Intelligence (IJCAI-17) describes our novel Description Logic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$, its decidability, and associated reasoning tasks and their complexity.

- Sawangphol W., Li Y.-F., & Tack G. (2017), Revisiting Description Logics with Concrete Domains and Aggregation. Submitted to: the 26th International Joint Conference on Artificial Intelligence (IJCAI-17). Melbourne, Australia. 19 August – 25 August 2017

Chapter 2

Description Logics

In this chapter, the essential background of Description Logics (DLs), which is the main domain of our research is provided. Firstly, the overview of the ideas underlying Description Logics and their use and relationship in knowledge representation is described. Secondly, the syntax and the semantics of a variety of Description Logics used in this thesis is presented. Thirdly, some main reasoning services are discussed. Finally, we close this chapter with a discussion on Description Logic families.

2.1 Knowledge Representation and Description Logic

Generally, *Knowledge Representation* (KR) is the approach to expressing knowledge in a computer-interpretable form about a specific domain. KR has played an important role in the fields of artificial intelligence. The fundamental goal of KR and reasoning is to represent knowledge and draw new conclusions from expressed knowledge.

Description Logics (DLs) are a successful family of knowledge representation languages, which provide semantics to model of relationships between objects in particular domains in a logical manner (Baader & Nutt, 2003). In Description Logics, many language constructs have been defined. For many years, researchers have studied the effects of defining and combining these language constructs on the complexity and the expressive power of the different Description Logics obtained. A wide range of Description Logics, which have low expressive power to high expressive power, are the results of their research. As their name indicates, DLs are logics equipped with formal logical semantics. The semantics allow humans and computer systems to exchange knowledge without ambiguity. In addition, DLs have been more important since the advent of the Semantic Web, where Description Logics play an important role as the main knowledge representation formalism, which provides the underpinnings for the Web Ontology Language (OWL) (Horrocks et al., 2003) and its extension OWL 2 (Grau et al., 2008). DLs are used to define three types of entities, which are *concepts* representing sets of individuals, *roles* representing binary relationship between individuals, and *individuals* representing specific objects, in a specific domain. Moreover, DLs consist of a set of statements, or *axioms*. Axioms are used to capture partial knowledge of a particular domain. There are three main groups of axioms. Terminological axioms (TBox) capture general knowledge about concepts. Assertional axioms (ABox) describe knowledge about atomic individuals, such as a relationship between two individuals via a role and membership of an individual in a concept. Role characteristics (RBox) describe knowledge about roles.

For example, an ontology about the domain of student and university relationships may use concepts such as $\mathsf{Student}$ and $\mathsf{University}$ to represent set of students and universities, a role such as $\mathsf{studyAt}$ to represent the (binary) relationship between students and universities, and individuals such as $\mathsf{wudhichart}$ and monash to represent the student and university respectively. The ontology may have TBox axioms such as $\mathsf{Student} \equiv \exists \mathsf{studyAt.University}$ to state that $\mathsf{Student}$ are those students that study at $\mathsf{University}$ and ABox axioms such as $\mathsf{Student(wudhichart)}$ to assert that Wudhichart is a $\mathsf{Student}$.

One of the important properties of Description Logics is the computation of inferences (a.k.a. reasoning). The formal semantics, which DLs are equipped, enables reasoning services to deduce implicit knowledge from explicit knowledge. There are four key reasoning services, which are *consistency*, *concept satisfiability*, *concept subsumption*, and *classification*. In a KR system, since it needs to ensure that it should always answer user queries in acceptable time, the main goal of Description Logics design has been to maintain a balance between expressivity and complexity of Description Logics. Therefore, most DLs developed are decidable. In general, for very expressive DLs, reasoning can be unfortunately very hard (NP-COMPLETE, PSPACE-COMPLETE, EXPTIME-COMPLETE and more) (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003).

For Description Logics, many language constructs have been developed to model concepts of domains, and objects and relationships among them. Different combinations of these language constructs yield different Description Logics. In addition, the combination of these language constructs can influence the decidability, the expressive power and the complexity of the reasoning problem. On the one hand, expressive DLs can express many phenomena but inference problems are of high complexity. On the other hand, weak DLs with low complexity may not be expressive enough to describe important concepts in some application domains. Therefore, the investigation to balance between the expressive power of Description Logic and the complexity of inference problems has been one of the most important topics in DL research.

2.2 Syntax, Notation and Semantics

In this section, we formally introduce syntax and semantics of some Description Logics (DLs).

An *ontology* $\mathcal{O}$ (or knowledge base) based on DLs consists of three components. The first one is the terminological component (TBox), which is the schema of the ontology including concept expressions and role expressions. The second component is the assertional component (ABox), which is an instantiation of the schema of the ontology. The third component is the role characteristic component (RBox) (Krötzsch, Simancik, & Horrocks, 2012), which describes properties of roles. Note that we only focus on ontologies that consists of TBox and ABox in this thesis. The presence of one or all of these three components also can effect the complexity of inference problems. In addition, the characteristics of TBox can influence the complexity of inference problems. For example, a cyclic or non-cyclic TBox may have different complexity.

Some well-known Description Logics are $\mathcal{EL}$ (Baader, 2003), $\mathcal{ALC}$ (Schmidt-Schauß & Smolka, 1991), $\mathcal{SHOIQ}$ (Horrocks & Sattler, 2005), and $\mathcal{SROIQ}$ (Horrocks et al., 2006).

2.2.1 Syntax and Notation

In this section, the syntax and notation of Description Logic are formally introduced.

The common language constructs that many Description Logics are equiped with are listed in Table 2.1. Formally, *concepts descriptions* in a Description Logic are defined through the fixed set of vocabulary N, which consists of non-empty sets N_A of *concept names (or atomic concept)*, N_R of *atomic roles*, and N_I of *individual names*. All complex concepts and roles are recursively defined through the language constructs in Table 2.1 In this research, we use the letter $A, B, C, C_i, D, D_i, E, ..$ to represent atomic concepts, whereas the letter $X, X_i, Y, Y_i, ..$ or the notation $\hat{C}, \hat{D}, ...$ are used to represent complex concept descriptions. In addition, we use the letters $R, S, ..$ to represent atomic roles, while the individual names are represented by the lowercase letters $a, b, c, ...$

Terminology axioms (TBox) $\mathcal{T}$ in Description Logic are a finite set of *concept axioms* and *role characteristics*, which are defined through axiom constructs. The axiom constructs are defined from the set of concepts and roles. The main *axiom constructs* of Description Logics are listed in Table 2.2. Note that a *concept equivalence* axiom $\hat{C} \equiv \hat{D}$ abbreviates the two concept inclusions $\hat{C} \sqsubseteq \hat{D}$ and $\hat{D} \sqsubseteq \hat{C}$. Similarly, a *role equivalence* axiom $R \equiv S$ is used. Axioms in Description Logic that involve individuals are namely *assertional axioms* (ABox). An ABox is a finite set of *concepts* and *role assertions.* Role inclusions and compositions are called *complex role inclusions.* Another type of axiom is called *role axioms*(RBox) (Krötzsch et al., 2012). A RBox presents *role characteristics*, i.e., the properties of roles. The *size* of $\mathcal{O}$, denoted as $|\mathcal{O}|$ is the number of axioms in $\mathcal{O}$. Note that the language construct listed in Table 2.1 and axioms listed in Table 2.2 are allowed in the Description Logic $\mathcal{SROIQ}$ (Horrocks et al., 2006). There are other concept constructs such as concrete domain, which will be introduce in Section 2.5. Concrete domain is one of the constructs that we focus on in this work.

Let us consider the types of TBox. In general, *concept inclusions*, defined in Table 2.2, are the main components of a TBox. Given a TBox $\mathcal{T}$, if there are no cyclic dependencies among concept descriptions or names in the axioms in $\mathcal{T}$, then $\mathcal{T}$ is called *acyclic* or *unfoldable* (Baader, Horrocks, & Sattler, 2008). In other words, $\mathcal{T}$ is said to be acyclic if concepts are neither defined by themselves nor other concepts that refer to them. In addition, only one *concept definition* $A \equiv \hat{C}$ or *primitive concept definition* $A \sqsubseteq \hat{C}$, which is a special type of concept inclusion can be defined, for some concepts A in $\mathcal{T}$. If a TBox $\mathcal{T}$ contains only concept definitions, such a TBox $\mathcal{T}$ is called *definitorial.* If this is not the case, $\mathcal{T}$ is said to be a *general TBox* (or *cyclic*), which is made up of a set of axioms of the form $\hat{C} \equiv \hat{D}$ and $\hat{C} \sqsubseteq \hat{D}$, where $\hat{C}$ and $\hat{D}$ are concept descriptions. This form of axioms $\hat{C} \sqsubseteq \hat{D}$ is called *general concept inclusions* (GCIs).

An ABox $\mathcal{A}$ consists of expressions of the forms $C(a)$ and $R(a, b)$, where a and b are individuals. $C(a)$ is called a *concept assertion* and $R(a, b)$ is called a *role assertion.*

2.2.2 Semantics

The semantics of an ontology $\mathcal{O}$ (including TBox and ABox) is defined in terms of *interpretations.* An interpretation $\mathcal{I}$ consists of a nonempty set $\Delta^{\mathcal{I}}$, namely the domain of $\mathcal{I}$, and an interpretation function $\cdot^{\mathcal{I}}$ that maps each atomic concept (or concept name) $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each atomic role (or role name) $R \in N_R$ to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual name $a \in N_I$ to the corresponding individual $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The extensions of $\cdot^{\mathcal{I}}$ to complex concepts and roles are defined in the *Semantics* column in Table 2.1, where $(a, b) \in R^{\mathcal{I}}$ means that an individual b in $\Delta^{\mathcal{I}}$ is a $R^{\mathcal{I}}$*-successor of* a.

If the corresponding condition in the *Semantics* column in Table 2.2 holds for an interpretation $\mathcal{I}$, $\mathcal{I}$ *satisfyies* an axiom $\propto$, denoted as $\mathcal{I} \models \propto$. $\mathcal{I}$ is a *model* of $\mathcal{O}$, denoted as $\mathcal{I} \models \mathcal{O}$, if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. An ontology is said to be *consistent* if there exists at least one model. Otherwise, it is *inconsistent.* A concept A is said to be *satisfiable* w.r.t. $\mathcal{O}$ if and only if there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that $A^{\mathcal{I}} \neq \emptyset$. Otherwise, A is *unsatisfiable* w.r.t. $\mathcal{O}$. An axiom $\propto$ is said to be a *consequence* of an ontology $\mathcal{O}$ (or $\mathcal{O}$ *entails* $\propto$, denoted as $\mathcal{O} \models \propto$) if every model of $\mathcal{O}$ satisfies $\propto$. A concept A is *subsumed* by B w.r.t. $\mathcal{O}$ if and only if $\mathcal{O} \models A \sqsubseteq B$. An individual a is an element of A w.r.t. $\mathcal{O}$ if and only if $\mathcal{O} \models A(a)$.

Table 2.1: The syntax and semantics of DL constructs.

Concepts	**Syntax**	**Semantics**
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
atomic concept	A	$A^{\mathcal{I}}$
concept negation	A	$A^{\mathcal{I}}$
conjunction	$\hat{C} \sqcap \hat{D}$	$\hat{C}^{\mathcal{I}} \cap \hat{D}^{\mathcal{I}}$
disjunction	$\hat{C} \sqcap \hat{D}$	$\hat{C}^{\mathcal{I}} \cap \hat{D}^{\mathcal{I}}$
existential restriction	$\exists R.\hat{C}$	$\{a \in \Delta^{\mathcal{I}} \mid$ there exists $b \in \Delta^{\mathcal{I}}$ such that $(a,b) \in R^{\mathcal{I}}$ and $y \in \hat{C}^{\mathcal{I}}\}$
universal restriction	$\forall R.\hat{C}$	$\{a \in \Delta^{\mathcal{I}} \mid$ for all $b \in \Delta^{\mathcal{I}}$ such that $(a,b) \in R^{\mathcal{I}}$ then $y \in \hat{C}^{\mathcal{I}}\}$
at-least restriction	$\geq nR.\hat{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \|\{b \in \hat{C}^{\mathcal{I}} \| (a,b) \in R^{\mathcal{I}}\| \geq n\}$
at-most restriction	$\leq mR.\hat{C}$	$\{a \in \Delta^{\mathcal{I}} \mid \|\{b \in \hat{C}^{\mathcal{I}} \| (a,b) \in R^{\mathcal{I}}\| \leq m\}$
local reflexivity	$\exists R.Self$	$\{a \in \Delta^{\mathcal{I}} \mid (a,a) \in R^{\mathcal{I}}\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
Roles	**Syntax**	**Semantics**
atomic role	R	$R^{I} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse role	R^{-}	$\{(a,b) \| (b,a) \in R^{I}\}$
Individuals	**Syntax**	**Semantics**
individual name	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$

Table 2.2: The syntax and semantics of some DL axioms.

TBox	**Syntax**	**Semantics**
concept inclusion	$\hat{C} \sqsubseteq \hat{D}$	$\hat{C}^{\mathcal{I}} \subseteq \hat{D}^{\mathcal{I}}$
concept equivalence	$\hat{C} \equiv \hat{D}$	$\hat{C}^{\mathcal{I}} = \hat{D}^{\mathcal{I}}$
ABox	**Syntax**	**Semantics**
concept assertion	$\hat{C}(a)$	$a^{\mathcal{I}} \in \hat{C}^{\mathcal{I}}$
role assertion	$R(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
individual equality	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
RBox	**Syntax**	**Semantics**
role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
role composition	$R_1 \circ R_2 \sqsubseteq S$	$(x,y) \in R_1^{\mathcal{I}} \wedge (y,z) \in R_2^{\mathcal{I}} \rightarrow (x,z) \in S^{\mathcal{I}}$
transitive role	transitive(R)	$(x,y) \in R^{\mathcal{I}} \wedge (y,z) \in R^{\mathcal{I}} \rightarrow (x,z) \in R^{\mathcal{I}}$
reflexive role	reflexive(R)	$(x,x) \in R^{\mathcal{I}}$ for all $x \in \Delta^{\mathcal{I}}$
irreflexive role	irreflexive(R)	$(x,x) \notin R^{\mathcal{I}}$ for all $x \in \Delta^{\mathcal{I}}$
symmetric role	symmetric(R)	$(x,y) \in R^{\mathcal{I}} \rightarrow (y,x) \in R^{\mathcal{I}}$
asymmetric role	asymmetric(R)	$(x,y) \in R^{\mathcal{I}} \rightarrow (y,x) \notin R^{\mathcal{I}}$
disjoint roles	disjoint(R,S)	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$

2.3 Reasoning Services

Reasoning services are one of the main purposes of Description Logic. Reasoning is the task that makes certain implicit knowledge explicit in an ontology (or knowledge base) (Baader & Nutt, 2003). As mentioned above, ontologies are given semantics by Description Logics.

Various reasoning services have been defined in Description Logics. Some of them are considered as fundamental of every DL. In general, core reasoning services are commonly supported by all state-of-the-art reasoners. These services are normally called *standard* reasoning services.

The main reasoning tasks described in this section are *concept satisfiability*, *concept subsumption*, *classification*, *consistency checking*, and *instance checking* (Baader & Nutt, 2003; Suntisrivaraporn, 2009; Turhan, 2010). The first two tasks are performed on specific concepts with respect to terminological axioms in a particular ontology. In contrast, the third and fourth tasks are performed on all concepts with respect to terminological axioms in the ontology. The last task is performed on assertional axioms.

We give the formal definition of these reasoning tasks. First of all, given ontology $\mathcal{O}$, The first four tasks are performed on $\mathcal{T}$ and the last task is performed on $\mathcal{A}$.

Concept Satisfiability

The *concept satisfiability* task is to ensure that a particular concept is satisfiable w.r.t $\mathcal{T}$, where $\mathcal{T}$ is consistent. A concept description $\hat{C}$ is satisfiable w.r.t $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty. In other words, there exists an individual $a \in \Delta^{\mathcal{I}}$, which is an instance of $\hat{C}$ ($a \in \hat{C}^{\mathcal{I}}$). Otherwise, concept description $\hat{C}$ is not satisfiable. Note that concept satisfiability is sometimes considered with empty TBox, i.e., the concept description alone is considered. In such a case, an arbitrary interpretation that makes $\hat{C}$ non-empty is considered. In addition, $\mathcal{T}$ is consistent if there exists a model $\mathcal{I}$ of all axioms in $\mathcal{T}$. Otherwise, $\mathcal{T}$ is not consistent. Nevertheless, this task is trivial for DLs that do not allow negative concept in particular $\mathcal{EL}$ DL family because every concept is satisfiable.

Concept Subsumption

Rather than concept satisfiability, *concept subsumption checking* is used to check whether some concept is more general than another one. A concept description $\hat{C}$ is subsumed by a concept description $\hat{D}$ w.r.t $\mathcal{T}$ if $\hat{C}^{\mathcal{I}} \subseteq \hat{D}^{\mathcal{I}}$ is true for all models $\mathcal{I}$ of $\mathcal{T}$. In this case, a concept description $\hat{C}$ is subsumed by a concept description $\hat{D}$ w.r.t $\mathcal{T}$ can be written by $\hat{C} \sqsubseteq_{\mathcal{T}} \hat{D}$ or $\mathcal{T} \models \hat{C} \sqsubseteq \hat{D}$. We call $\hat{C}$ a *subsumee*, *subconcept*, or *subclass*, and $\hat{D}$ a *subsumer*, *superconcept*, or *superclass*. In some DLs without negation such as $\mathcal{EL}$, concept satisfiability is not interesting. However, having the bottom concept in DLs, concept satisfiability can be reduced to concept subsumption checking. A concept description $\hat{C}$ is unsatisfiable w.r.t. $\mathcal{T}$ if and only if $\hat{C} \sqsubseteq_{\mathcal{T}} \bot$.

Classification

The *classification* task is to determine subsumption relationship between all pairs of concept names occurring in $\mathcal{T}$. For example, for all $A, B \in N_C$, where N_C is the set of concept names in $\mathcal{T}$, the classification task determines whether $A \sqsubseteq_{\mathcal{T}} B$ or $B \sqsubseteq_{\mathcal{T}} A$.

Consistency Checking

The *consistency checking* task is used to determine whether a given ontology is consistent. A TBox $\mathcal{T}$ is consistent if and only if there exists a model $\mathcal{I}$ for $\mathcal{T}$. If this is not the case, $\mathcal{T}$ is said to be inconsistent. An interpretation $\mathcal{I}$ is a model for a given $\mathcal{T}$ if and only if $\mathcal{I}$ satisfies all axioms in $\mathcal{T}$.

Instance Checking

The *instance checking* is the task to decide whether a given individual is an instance of a given concept and a given pair of individuals is an instance of a given role. Formally, given a TBox $\mathcal{T}$, a concept A, and a role R in $\mathcal{T}$, a given individual a is an *instance* of A if and only if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. Similarly, a given pair of individuals (a, b) is an *instance* of R if and only if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.

2.4 Description Logic Families

Description Logics are named by the language constructs and axioms they allow. For example, the core Description Logic $\mathcal{ALC}$ (Schmidt-Schauß & Smolka, 1991) is a fragment of $\mathcal{SROIQ}$. $\mathcal{ALC}$ only allows the language constructs top concept, bottom concept, concept name, conjunction, disjunction, existential restriction, universal restriction, and concept inclusion axioms. The syntax of $\mathcal{ALC}$ concepts is defined as follows:

$$\hat{C} := \top \mid \bot \mid A \mid \neg\hat{C} \mid \hat{C} \sqcap \hat{D} \mid \hat{C} \sqcup \hat{D} \mid \exists R.\hat{C} \mid \forall R.\hat{C}$$

The Description Logic $\mathcal{ALC}$ extended by transitive roles is denoted by the letter $\mathcal{S}$ (Horrocks, Sattler, & Tobies, 1999). Generally, the names of Description Logic with additional letters identify a particular language constructs or axioms allowed in those Description Logics, such as unqualified number restrictions denoted by $\mathcal{N}$, qualified number restrictions denoted by $\mathcal{Q}$, role inclusions denoted by $\mathcal{H}$, inverse roles denoted by $\mathcal{I}$, and nominals denoted by $\mathcal{O}$. For example, the DL $\mathcal{ALCQ}$ (Hollunder & Baader, 1991) extends $\mathcal{ALC}$ with qualified number restrictions. Commonly, the combination of complex role inclusions, role characteristics, and local reflexivity is denoted by the letter $\mathcal{R}$. Table 2.3 summarises this naming scheme.

Table 2.3: Summary of the Description Logic naming scheme.

Notation	**Description**
$\mathcal{A}$	Atomic concept
$\mathcal{AL}$	Top concept, Bottom concept, conjunction, universal restriction
$\mathcal{U}$	Disjunction
$\mathcal{C}$	Negation
$\mathcal{E}$	existential restriction
$\mathcal{S}$	$\mathcal{ALC}$ extended with transitive roles
$\mathcal{N}$	Unqualified number restriction
$\mathcal{Q}$	Qualified number restriction
$\mathcal{O}$	Nominals
$(\mathbf{D})$	Data types
$\mathcal{D}$	Concrete domains
$\mathcal{H}$	Role hierarchies
$\mathcal{I}$	Inverse roles
$\mathcal{R}$	Complex role inclusions, i.e., $R \circ S \sqsubseteq R$

Next, we provide a formal introduction to the syntax and semantics of some Description Logics (DLs), starting from a simple DL $\mathcal{EL}$ and a well-known DL $\mathcal{ALC}$. Finally, two Description Logics $\mathcal{ALC}$ and $\mathcal{EL}$ extended with concrete domain and aggregations are formally presented.

2.4.1 Light-Weight Description Logics

Since the reasoning services of most of Description Logics are intractable, a lot of research has been investigated to define tractable Description Logics. Note that tractability means that the reasoning complexity is PTime-complete. This is in contrast to the trend of the last two decade, in which most research has focused on investigating increasing expressive power of Description Logics.

The Description Logic $\mathcal{EL}$ was designed to be tractable (Baader, 2003), while being expressive enough to represent knowledge in several large and widely-used biomedical ontologies that mainly consist of terminological axioms, such as GALEN (Rector, Rogers, & Pole, 1996), Gene ontology (GO, see Figure 2.1) (Ashburner et al., 2000), and SNOMED CT (Stearns et al., 2001). Besides the Description Logic $\mathcal{EL}$, many other extensions of $\mathcal{EL}$ have been investigated such as $\mathcal{EL}^+$ and $\mathcal{EL}^{++}$ (Baader, Brandt, & Lutz, 2005; Baader, Lutz, & Brandt, 2008).

$$\begin{aligned}
\mathsf{DomainCategory} &\sqsubseteq \mathsf{TopCategory} \\
\mathsf{GeneralisedStructure} &\sqsubseteq \mathsf{DomainCategory} \\
\mathsf{AbstractStructure} &\sqsubseteq \mathsf{GeneralisedStructure} \\
\mathsf{DiabetogenicStructure} &\equiv \mathsf{GeneralisedStructure} \sqcap \exists\ \mathsf{IsCausallyLinkedTo.Diabetes}
\end{aligned}$$

Figure 2.1: A fragment of Gene Ontology (GO).

In more details, the *concept descriptions* in $\mathcal{EL}$ are defined through the constructors in the upper half of Table 2.4. Let $A, B, ...$ represent *concept names*, $\hat{C}, \hat{D}, \ldots$ represent (anonymous) concept descriptions, and $R, S, ...$ represent *role names.* We use N_C to represent a non-empty set of concept names and N_R to represent a non-empty set of role names. An $\mathcal{EL}$ TBox (*terminology box* or *ontoloogy*) $\mathcal{T}$ is a finite set of axioms as defined in the lower part of Table 2.4.

Table 2.4: The syntax and semantics of $\mathcal{EL}$.

Concepts	**Syntax**	**Semantics**
top concept	$\top$	$\Delta^\mathcal{I}$
atomic concept	A	$A^\mathcal{I}$
conjunction	$\hat{C} \sqcap \hat{D}$	$\hat{C}^\mathcal{I} \cap \hat{D}^\mathcal{I}$
existential restriction	$\exists R.\hat{C}$	$\{x \in \Delta^\mathcal{I} \mid$ there exists $y \in \Delta^\mathcal{I}$ such that $(x, y) \in R^\mathcal{I}$ and $y \in \hat{C}^\mathcal{I}\}$
Axioms	**Syntax**	**Semantics**
concept inclusion (subsumption)	$\hat{C} \sqsubseteq \hat{D}$	$\hat{C}^\mathcal{I} \subseteq \hat{D}^\mathcal{I}$
concept equivalence	$\hat{C} \equiv \hat{D}$	$\hat{C}^\mathcal{I} = \hat{D}^\mathcal{I}$

Importantly, concept subsumption and classification w.r.t. general TBoxes of $\mathcal{EL}$ can be performed in *polynomial time* (Baader, Brandt, & Lutz, 2005; Baader, Lutz, & Brandt, 2008). The other reasoning services, concept satisfiability and consistency checking, can be polynomially reduced to each others. However, concept satisfiability and consistency checking are not interesting for $\mathcal{EL}$ since $\mathcal{EL}$ does not allow the language constructs, which can cause logical contradictions.

An $\mathcal{EL}$ Normal Form

Given an $\mathcal{EL}$ TBox $\mathcal{T}$, we use $BN_{\mathcal{T}}$ to represent the set of *basic concept* of $\mathcal{T}$. $BN_{\mathcal{T}}$ is the smallest set of concepts containing: (i) the top concept ($\top$); (ii) all concept names A of $\mathcal{T}$ in N_C. Then we restrict our attention to those $\mathcal{EL}$ TBoxes in which all axioms are in the following normal forms:

$$\begin{array}{rl} A \sqsubseteq B & \\ A_1 \sqcap A_2 \sqsubseteq B & \qquad A \sqsubseteq B_1 \sqcap B_2 \\ \exists R.A \sqsubseteq B & \qquad A \sqsubseteq \exists R.B \end{array}$$

where $A, A_1, A_2, B, B_1, B_2 \in BN_{\mathcal{T}}, R \in N_R$.

Any given TBox $\mathcal{T}$ can be transformed into an equivalent normalised TBox $\mathcal{T}'$. A TBox $\mathcal{T}'$ is a *conservative extension* of the TBox $\mathcal{T}$ if every model of $\mathcal{T}'$ is a model of $\mathcal{T}$ and every model of $\mathcal{T}$ can be extended to a model of $\mathcal{T}'$ by defining the interpretation for the additional concept names. The normalisation of $\mathcal{T}$ can be done in linear time by applying exhaustively the following normalisation rules (NR) (Suntisrivaraporn, 2009):

$$\hat{C} \equiv \hat{D} \rightsquigarrow \hat{C} \sqsubseteq \hat{D}, \hat{D} \sqsubseteq \hat{C} \tag{NR1}$$

$$\hat{C} \sqsubseteq \hat{D} \rightsquigarrow \hat{C} \sqsubseteq E, E \sqsubseteq F, F \sqsubseteq \hat{D} \tag{NR2}$$

$$A \sqcap \hat{C} \sqsubseteq B \rightsquigarrow A \sqcap E \sqsubseteq B, \hat{C} \sqsubseteq E \tag{NR3}$$

$$A \sqsubseteq B \sqcap \hat{D} \rightsquigarrow A \sqsubseteq B \sqcap E, E \sqsubseteq \hat{D} \tag{NR4}$$

$$\exists R.\hat{C} \sqsubseteq B \rightsquigarrow \exists R.E \sqsubseteq B, \hat{C} \sqsubseteq E \tag{NR5}$$

$$A \sqsubseteq \exists R.\hat{D} \rightsquigarrow A \sqsubseteq \exists R.E, E \sqsubseteq \hat{D} \tag{NR6}$$

where $A, B \in BN_{\mathcal{T}}, \hat{C}, \hat{D} \notin BN_{\mathcal{T}}$, and E, F are fresh concept names introduced to define complex concept descriptions.

Additionally, we can transform n-ary conjunctions of concept names, which is represented as $\sqcap_i A_i$ by applying the following equivalence transformation to the axiom:

$$A \sqsubseteq B_1 \sqcap ... \sqcap B_n \rightsquigarrow A \sqsubseteq B_1, ..., A \sqsubseteq B_n \tag{NR7}$$

Therefore, we can remove the right-hand side conjunctions. The following is the resulting normal form:

$$\begin{array}{ll} A \sqsubseteq B & \qquad \sqcap_i A_i \sqsubseteq B \\ A \sqsubseteq \exists R.B & \qquad \exists R.A \sqsubseteq B \end{array}$$

where A, A_i, B are concept names.

$\mathcal{EL}$ can be extended in several ways ways. On one hand, it has been extended by adding new concept constructs to obtain more expressivity in describing the abstract domain (concepts). For example, the resulting DL is $\mathcal{EL}^{++}$ (Baader, Brandt, & Lutz, 2005; Baader, Lutz, & Brandt, 2008). We use the well-known DL $\mathcal{ALC}$ to describe such extensions in Section 2.4.2. On the other hand, it can be extended to obtain more expressivity by adding support for concrete domains (e.g., natural numbers). We illustrate such an extension with DL $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ in Chapter 6.

2.4.2 Core Description Logic $\mathcal{ALC}$

One of the most well-known descriptions logics is $\mathcal{ALC}$ (Schmidt-Schauß & Smolka, 1991). While it predates $\mathcal{EL}$, logically it can be seen as an extension of it. In addition to the constructs supported by $\mathcal{EL}$, $\mathcal{ALC}$ contains the bottom concept ($\bot$), concept disjunctions ($\sqcup$), universal restrictions ($\forall$), and concept negation ($\neg$). The concept descriptions in $\mathcal{ALC}$ are defined through the concept constructs listed in Table 2.5. In fact, $\mathcal{ALC}$ provide complete propositional reasoning since it allows negation, disjunction, and conjunction. Additionally, $\mathcal{ALC}$ offers reasoning about individuals , through existential and universal restrictions. Unfortunately, the complexity of reasoning services of $\mathcal{ALC}$ is not tractable. The concept satisfiability and concept subsumption of $\mathcal{ALC}$ w.r.t. empty and acyclic TBoxes are PSPACE-COMPLETE (Schmidt-Schauß & Smolka, 1991). The complexity of these reasoning services becomes EXPTIME-COMPLETE for general TBoxes (Baader et al., 2003).

The semantics of $\mathcal{ALC}$ is defined the same as the semantics of $\mathcal{EL}$ presented in Section 2.4.1.

Table 2.5: The syntax and semantics of $\mathcal{ALC}$.

Concepts	**Syntax**	**Semantics**
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
atomic concept	A	$A^{\mathcal{I}}$
concept negation	$\neg\hat{C}$	$\Delta^{\mathcal{I}} \setminus \hat{C}^{\mathcal{I}}$
conjunction	$\hat{C} \sqcap \hat{D}$	$\hat{C}^{\mathcal{I}} \cap \hat{D}^{\mathcal{I}}$
disjunction	$\hat{C} \sqcup \hat{D}$	$\hat{C}^{\mathcal{I}} \cup \hat{D}^{\mathcal{I}}$
existential restriction	$\exists R.\hat{C}$	$\{a \in \Delta^{\mathcal{I}} \mid$ there exists $b \in \Delta^{\mathcal{I}}$ such that $(a,b) \in R^{\mathcal{I}}$ and $b \in \hat{C}^{\mathcal{I}}\}$
universal restriction	$\forall R.\hat{C}$	$\{a \in \Delta^{\mathcal{I}} \mid$ for all $b \in \Delta^{\mathcal{I}}$ such that $(a,b) \in R^{\mathcal{I}}$ then $b \in \hat{C}^{\mathcal{I}}\}$
Axioms	**Syntax**	**Semantics**
concept inclusion (subsumption)	$\hat{C} \sqsubseteq \hat{D}$	$\hat{C}^{\mathcal{I}} \subseteq \hat{D}^{\mathcal{I}}$
concept equivalence	$\hat{C} \equiv \hat{D}$	$\hat{C}^{\mathcal{I}} = \hat{D}^{\mathcal{I}}$

An $\mathcal{ALC}$ Normal Form

Wlog., we assume that all the $\mathcal{ALC}$ concept description is in *negative normal form* (NNF).

Definition 2.4.1. (Negation Normal Form (NNF)) A concept description is said to be in NNF if the negation ($\neg$) is only in front of concept names. NNF of concept descriptions can be obtained by pushing negations into concept descriptions (Horrocks, 2003).

From a concept description $\hat{C}$, it is possible to obtain an equivalent concept description in NNF by applying the following transformations (Schmidt-Schauß & Smolka, 1991). Note that the conversion to NNF is polynomial.

$$\neg\neg\hat{C} \rightsquigarrow \hat{C} \tag{TR1}$$
$$\neg\top \rightsquigarrow \bot \tag{TR2}$$
$$\neg\bot \rightsquigarrow \top \tag{TR3}$$
$$\neg(\hat{C} \sqcap \hat{D}) \rightsquigarrow \neg\hat{C} \sqcup \neg\hat{D} \tag{TR4}$$
$$\neg(\hat{C} \sqcup \hat{D}) \rightsquigarrow \neg\hat{C} \sqcap \neg\hat{D} \tag{TR5}$$
$$\neg\exists R.\hat{C} \rightsquigarrow \forall R.\neg\hat{C} \tag{TR6}$$
$$\neg\forall R.\hat{C} \rightsquigarrow \exists R.\neg\hat{C} \tag{TR7}$$

Then we restrict our attention to those $\mathcal{ALC}$ TBoxes in normal form, where all axioms are of the following forms:

$$\begin{array}{rl}
A \sqsubseteq B & \\
A_1 \sqcap A_2 \sqsubseteq B & A \sqsubseteq B_1 \sqcap B_2 \\
A_1 \sqcup A_2 \sqsubseteq B & A \sqsubseteq B_1 \sqcup B_2 \\
\exists R.A \sqsubseteq B & A \sqsubseteq \exists R.B \\
\forall R.A \sqsubseteq B & A \sqsubseteq \forall R.B
\end{array}$$

where $A, A_1, A_2, B, B_1, B_2 \in BN_{\mathcal{T}}, R \in N_R$.

The normalisation of $\mathcal{ALC}$ TBox $\mathcal{T}$ can be done in linear time by applying (NR1) - (NR7) above and the following additional normalisation rules:

$$A \sqcup \hat{C} \sqsubseteq B \rightsquigarrow A \sqcup E \sqsubseteq B, \hat{C} \sqsubseteq E \tag{NR8}$$
$$A \sqsubseteq B \sqcup \hat{D} \rightsquigarrow A \sqsubseteq B \sqcup E, E \sqsubseteq \hat{D} \tag{NR9}$$
$$\forall R.\hat{C} \sqsubseteq B \rightsquigarrow \forall R.E \sqsubseteq B, \hat{C} \sqsubseteq E \tag{NR10}$$
$$A \sqsubseteq \forall R.\hat{D} \rightsquigarrow A \sqsubseteq \forall R.E, E \sqsubseteq \hat{D} \tag{NR11}$$

where $A, B \in BN_{\mathcal{T}}, \hat{C}, \hat{D} \notin BN_{\mathcal{T}}$, and E, F are fresh concept names introduced to define complex concept description.

In addition, we can transform n-ary disjunctions of concept names, which is represented as $\sqcup_i A_i$ by applying the following equivalence transformation to the axiom:

$$A_1 \sqcup ... \sqcup A_n \sqsubseteq B \rightsquigarrow A_1 \sqsubseteq B, ..., A_n \sqsubseteq B \tag{NR12}$$

Therefore, we can remove the left-hand side disjunctions. The following is the resulting normal form:

$$\begin{array}{rr}
A \sqsubseteq B & \\
\sqcap_i A_i \sqsubseteq B & A \sqsubseteq \sqcup_i B_i \\
A \sqsubseteq \exists R.B & \exists R.A \sqsubseteq B \\
A \sqsubseteq \forall R.B & \forall R.A \sqsubseteq B
\end{array}$$

where A, A_i, B, B_i are concept names.

2.5 Description Logics with Concrete Domains and Aggregations

As can be seen, most of the Description Logics have been developed to improve expressivity for defining knowledge on the abstract logical level. In many applications, knowledge needs to be defined over *concrete domains* and *predicates* in domains. Concrete domains and predicates first were introduced as an extension of $\mathcal{ALC}$, where concrete domains are integers, reals, sets of temporal intervals, or sets of spatial regions, and predicates include equality, temporal overlapping, and spatial disconnectedness, resulting the Description Logic $\mathcal{ALC}(\mathcal{D})$ (Baader & Hanschke, 1991). The integration of concrete domains in Description Logics has been found interesting in a number of applications such as mechanical engineering (Baader & Hanschke, 1992), reasoning with physical laws (Kamp & Wache, 1996), and temporal and spatial reasoning (Haarslev, Lutz, & Möller, 1999). Moreover, some extensions, such as feature (dis) agreement (Lutz, 2002) and database-like key constraint (Lutz et al., 2003), of Description Logics with concrete domains have been studied.

In addition, aggregation functions (e.g., sum and count) are fundamental and useful mechanism available in database systems. Aggregations was first introduced as an extension of $\mathcal{ALC}(\mathcal{D})$ (Baader & Hanschke, 1991), resulting in the Description Logic $\mathcal{ALC}(\Sigma)$ (Baader & Sattler, 2003).

2.5.1 Concrete Domains

We first illustrate the use of *concrete domains* in knowledge representation with an example. Using concrete domains constructs presented in (Baader & Hanschke, 1992), we can define a treadmills fitness machine as follows:

$$\begin{aligned}&\mathsf{Treadmill} \sqcap \exists \mathsf{runningsurface.RunningSurface} \sqcap \\ &=_{120} .(\mathsf{powerrequired}) \sqcap\ > .(\mathsf{width}, (\mathsf{runningsurface\ width}))\end{aligned}$$

This concept expresses a treadmill fitness machine with a required power of 120 watts and the width of the machine is greater than the width of its running surface. As can be seen, the third and fourth conjuncts are represented using concrete domain constructs. Precisely, runningsurface is an *abstract feature*, which is interpreted as a partial function from the abstract domain $\Delta^{\mathcal{I}}$ to the abstract domain $\Delta^{\mathcal{I}}$, whereas powerrequired and width are called *concrete features*, which are interpreted as a partial function from the abstract domain $\Delta^{\mathcal{I}}$ to the concrete domain (e.g., the natural numbers). The expression (runningsurface width) is a chain of features, which represents the width of the runningsurface-successor and $>$ is a predicate from the concrete domain. Let us now present concrete domain formally.

Definition 2.5.1. A concrete domain $\mathcal{D}$ is a pair of $(\Delta^{\mathcal{D}},\Phi^{\mathcal{D}})$, where $\Delta^{\mathcal{D}}$ is a domain and $\Phi^{\mathcal{D}}$ is a set of predicate names. For each predicate name P in $\Phi^{\mathcal{D}}$, P is associated with an arity n and an n-ary predicate $P^{\mathcal{D}} \subseteq \Delta^{\mathcal{D}}_n$. Let V denote a set of variables. A predicate conjunction is of the form

$$c = \bigwedge_{i<k} P_i(x_1^{(i)}, ..., x_{n_i}^{(i)})$$

where P_i is an n_i-ary predicate name for $i < k$, k is a number of predicate names, and $x_j^{(i)}$ are variables in V. Such a conjunction is said to be *consistent* if and only if there exists an assignment of elements in $\Delta^{\mathcal{D}}$ to the variables such that the conjunction is true in $\mathcal{D}$. Such an assignment is called a *solution* of c.

Hereafter, we use $\mathcal{D}$-*consistency* to refer to the consistency of finite conjunction of predicates from concrete domain $\mathcal{D}$. In the following, we introduce $\mathcal{ALC}(\mathcal{D})$, the well-known Description Logic $\mathcal{ALC}$ extended with concrete domain $\mathcal{D}$ (Baader & Hanschke, 1991).

The syntax of $\mathcal{ALC}(\mathcal{D})$ is defined as follows. Let N_C be a set of concepts, N_R be a set of roles, and N_F is a set of *features.* N_C, N_R, and N_F are disjoint. A *feature chain* u is a non-empty sequence of feature f_i, i.e., $u = f_1, ..., f_m$. Let $\mathcal{D}$ be a concrete domain. $\mathcal{ALC}(\mathcal{D})$ is extended from $\mathcal{ALC}$ by allowing the use of the concrete domain construct $P(u_1, ..., u_n)$, where $u_1, ..., u_m$ are feature chains and $P \in \Phi^{\mathcal{D}}$ is a predicate with arity n. In addition, $\mathcal{ALC}(\mathcal{D})$ disallows top concept ($\top$) and bottom concept ($\bot$). Then we introduce the semantics of the additional construct of concrete domain.

The semantics of $\mathcal{ALC}(\mathcal{D})$ is defined as follows. The semantics of the basic components (i.e., concepts and roles) of $\mathcal{ALC}(\mathcal{D})$ is the same as those of $\mathcal{ALC}$. We only present the additional components, which are feature and concrete domain construct $P(u_1, ..., u_n)$ here. Every feature f is mapped to a partial function $f^{\mathcal{I}}$ from $\Delta^{\mathcal{I}}$ to $\Delta^{\mathcal{I}} \cup \Delta^{\mathcal{D}}$. The semantic $P(u_1, ..., u_n)^{\mathcal{I}}$ of the concrete domain construct $P(u_1, ..., u_n)$ is $\{a \in \Delta^{\mathcal{I}} \mid \exists x_i \text{ such that } u_i^{\mathcal{I}}(a) = x_i \text{ and } (x_i, ..., x_n) \in P^{\mathcal{D}}\}$. Note that existential restrictions and universal restrictions are also defined for features. For a chain of features $u = f_1 f_2 ... f_n$, we can use notations $\exists u.C$, which is abbreviation for $\exists f_1.\exists f_2....\exists f_n.C$, and $\forall u.C$, which is abbreviation for $\forall f_1.\forall f_2....\forall f_n.C$. The semantics of these are defined in the same way as those of existential quantification and universal quantification for roles in Table 2.1.

The satisfiability of $\mathcal{ALC}(\mathcal{D})$ is decidable if the concrete domain $\mathcal{D}$ is admissible (Baader & Hanschke, 1991). The admissibility of concrete domain is essential for devising general decision procedures. Let us formally introduce the admissibility of concrete domains.

Definition 2.5.2. A concrete domain is called *admissible* if and only if

1. its set of predicate names $\Phi^{\mathcal{D}}$ is closed under negation, i.e., $\Phi^{\mathcal{D}}$ contains an n-ary predicate name $\bar{P}$ such that $\bar{P}^{\mathcal{D}} = \Delta_n^{\mathcal{D}} \setminus P^{\mathcal{D}}$ for each n-ary predicate name P in $\Phi^{\mathcal{D}}$,

2. its set of predicate names $\Phi^{\mathcal{D}}$ contains a name for $\Delta^{\mathcal{D}}$,

3. the satisfiability of finite conjunctions over $\Phi^{\mathcal{D}}$, i.e., the satisfiability of the predicate conjunctions in Definition 2.5.1, is decidable.

The Description Logic $\mathcal{ALC}(\mathcal{D})$ is the first carefully considered treatment of concrete domain. There are a number of studies of Description Logic with concrete domain as follows:

1. A Description Logic with expressive concrete domain CTL was developed to overcome the shortcomings of DL system w.r.t. system of (in)equality (Kamp & Wache, 1996). CTL was successfully developed by a combination of Description Logic as base logic and Constraint Logic Programming (CLP) as decision procedures. This logic is useful as a powerful tool for reasoning about technical devices (Kamp & Wache, 1996).

2. Many studies focus on combining general TBoxes with restricted forms of concrete domains. The Description Logic $\mathcal{SHOQ}$(D) allows only concrete datatypes and unary concrete domain predicates in a very expressive Description Logic (Horrocks & Sattler, 2001). For example, $\mathcal{SHOQ}$(D) can express the concept $\mathsf{Student} \sqcap$

$\exists$ scores.(min70), where (min70) is a concrete datatype. In addition, (min70) can be interpreted as a unary concrete domain predicate $P_{\geq 70}$. In (Pan, 2007), the Description Logic $\mathcal{SHOQ}(\mathcal{G})$ was mainly developed to support user-defined data type and data type predicates in ontology applications. For example, the due date of assignment on 13/03/2017 can be defined as *earlier than 13/03/2017*, where *earlier than 13/03/2017* is a user-predefined data type predicate. In (Haarslev, Möller, & Wessel, 2001), the combination of concrete domain without feature chains and Description Logics with role hierarchies, transitive roles and qualified number restrictions was studied resulting in the Description Logic $\mathcal{ALCNH}_{R^+}(\mathcal{D})^-$. However, predicates of arbitrary arity are admitted The resulting logic was claimed that it is useful for solving configuration problems. In (Lutz & Milicic, 2007), the combination of Description Logic with concrete domain and general TBoxes was studied resulting in the Description Logic $\mathcal{ALC}(\mathcal{C})$. The concrete domain used in this research is based on the RCC-8 relation, which is used to describe relations between regions in topological spaces. As can be seen, the above mentioned approaches basically restrict the expressivity of concrete domains to combine them with very expressive Description Logics.

3. A number of studies focus on combining expressive Description Logics with unrestricted concrete domains. The Description Logic $\mathcal{ALCF}(\mathcal{D})$ is an extension of $\mathcal{ALC}(\mathcal{D})$ by adding feature (dis)agreements (Lutz, 2002). The complexity of $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability is PSpace-complete if $\mathcal{D}$-consistency is PSpace-complete (Lutz, 2002). This logic is said to be useful in modelling for the application domain of disaster management (Kullmann, de Beuvron, & Rousselot, 2000). In addition, the Description Logic with concrete domain was extended by one of database-like constraints, concrete key constraint, namely $\mathcal{ALCK}(\mathcal{D})$ (Lutz et al., 2003). With this key constraint, it allows to express a uniqueness of an abject such as id of students. It turned out that integrating concrete key constraints to Description Logics with concrete domain can dramatically increase the complexity of reasoning and may lead to undecidability. Therefore, the use of concrete key constraints is restricted by allowing only Boolean combinations of concept names in concrete key constraint to retain decidability. The complexity of satisfiability of $\mathcal{ALCK}(\mathcal{D})$-concept is NExpTime-complete for very simple concrete domains such as bit vectors. In addition to concrete key constraints, functional dependencies were studied as an extension of Description Logic with concrete domain, namely $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ (Lutz & Milicic, 2004). Functional dependencies allow to express that a set of properties can decide the value of a property. For example, all employees with the same department id work in the same department. It turns out that the Description Logic with concrete domain and full functional dependencies lead to undecidability. To retain decidability, functional dependencies are restricted, where the concepts inside functional dependencies cannot have a subconcept defined by concrete domain constructs. The complexity of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$-concept satisfiability is NExpTime-complete for a simple concrete domain, where $\{0, 1\} \in \Delta^{\mathcal{D}}$ and only unary predicates are considered.

A summary of these studies is presented in Table 2.6.

2.5.2 Aggregations in Description Logics

We firstly demonstrate the use of *aggregations* in knowledge representation with Description Logics, using the language constructs presented in (Baader & Sattler, 2003). For example, we can define a concept of healthy person, where we compare calories taken and

Table 2.6: Summary of some Description Logics with concrete domains.

Description Logics	**Studies**
CTL (Kamp & Wache, 1996)	Combination of Description Logic as base logic and Constraint Logic Programming (CLP)
$\mathcal{SHOQ}$(D) (Horrocks & Sattler, 2001)	Concrete datatypes and unary concrete domain predicates
$\mathcal{SHOQ}(\mathcal{G})$ (Pan, 2007)	User-defined data type and data type predicates
$\mathcal{ALCNH}_{R^+}(\mathcal{D})^-$ (Haarslev et al., 2001)	The combination of concrete domain without feature chains and Description Logics with role hierarchies, transitive roles and qualified number restrictions
$\mathcal{ALC}(\mathcal{C})$ (Lutz & Milicic, 2007)	The combination of Description Logic with concrete domain, the RCC-8 relation, and general TBoxes
$\mathcal{ALCF}(\mathcal{D})$ (Lutz, 2002)	Description Logic with concrete domain and feature (dis)agreements
$\mathcal{ALCK}(\mathcal{D})$ (Lutz et al., 2003)	Description Logic with concrete domain and concrete key constraint
$\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ (Lutz & Milicic, 2004)	Description Logic with concrete domain and functional dependencies

calories burnt for each week as follows:

$$\exists \mathsf{week}. \leq (\mathsf{sum}(\mathsf{day} \circ \mathsf{caloriestaken}), \mathsf{sum}(\mathsf{day} \circ \mathsf{caloriesburnt}))$$

Intuitively, this concept expresses a healthy person who takes calories less than calories burnt in some weeks. The aggregations are used to sum up weekly calories consumed and burnt. More precisely, week and day are role names, while caloriestaken and caloriesburnt are concrete features. The aggregation $\mathsf{sum}(\mathsf{day} \circ \mathsf{caloriestaken})$ expresses a mapping individual related to the sum over all caloriestaken-values over day.

For example, in Figure 2.2, given a person, Wud, who takes calories less than or equal to calories that he burns everyday. Calories taken and burnt of a whole week can be known by considering the sum over all days of that week. Rectangle boxes represent individuals in the abstract domain, and circles represent individuals in a concrete domain, which is natural numbers in this case. The green rectangle illustrates the sum of concrete domain individuals for calories taken. The orange rectangle illustrates the sum of concrete domain individuals for calories burnt.

In order to describe the idea of aggregation and introduce the Description Logic $\mathcal{ALC}(\Sigma)$, we need to introduce the notion of *multisets*. In a multiset, an individual can appear more than once. For example, the multiset $[\![10]\!]$ is different from the multiset $[\![10, 10, 10]\!]$. Note that we use the symbol "$[\![]\!]$" to represent multisets for the rest of this thesis.

Definition 2.5.3. (Multisets) (Baader & Sattler, 2003) Let S be a set. A *multiset* M over S is a mapping $M : S \to \mathbb{N}$, where $M(s)$ represents the number of occurrences of s in M, where $s \in S$. The set of all multisets of S is denoted by $MS(S)$. A multiset M over S is said to be finite if and only if $\{s \mid M(s) \neq 0\}$ is a finite set.

For multisets M, M' over S, $M \subseteq M'$ is used to represent $M(s) \leq M'(s)$ for each $s \in S$, and $s \in M$ is used to represent $M(s) \geq 1$. $M' \setminus M$ is used to denote the multiset with $(M' \setminus M)(s) := M'(s) - M(s)$ for all $s \in S$.

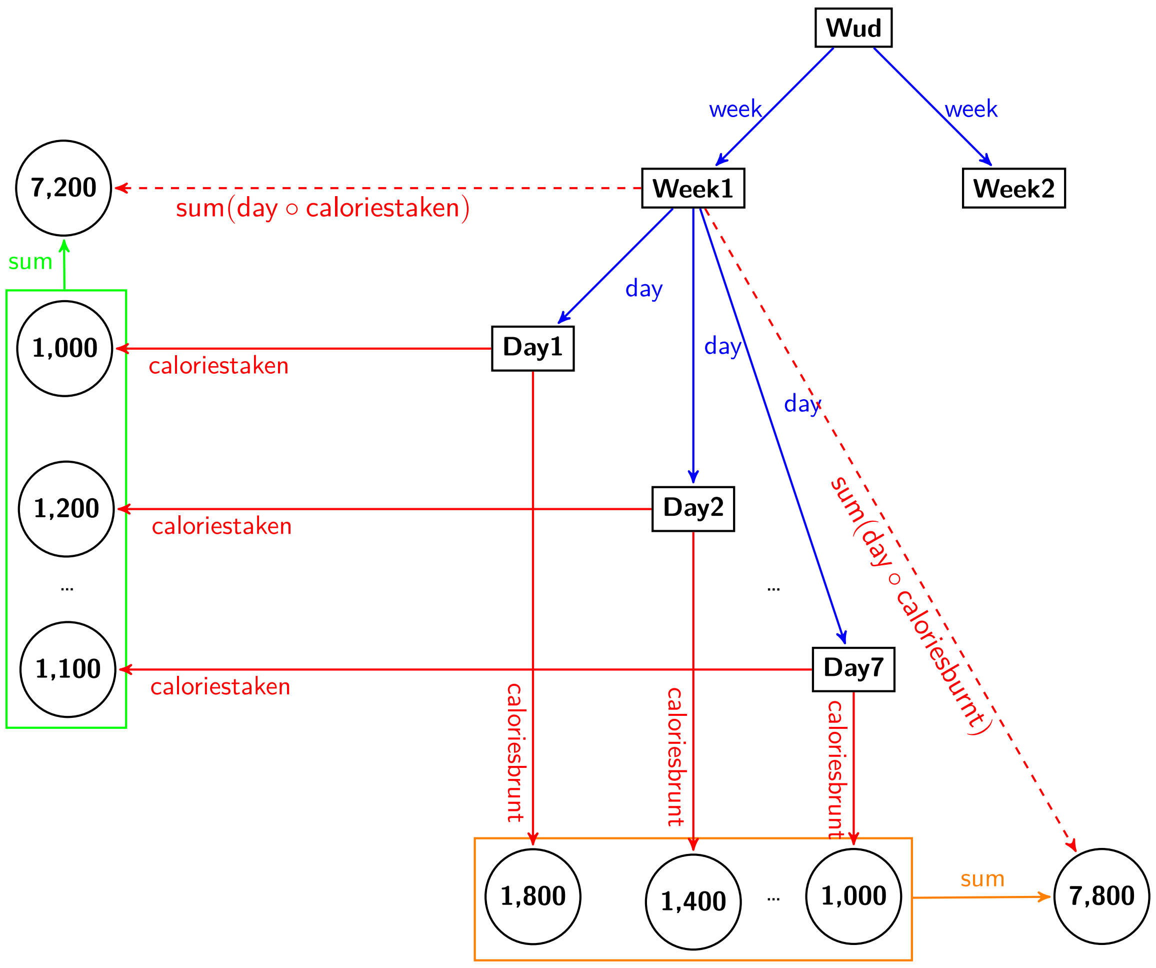


Figure 2.2: Example of aggregation.

In order to define aggregations, the notion of concrete domain and concrete features in concrete domains construct are extended as follows:

Definition 2.5.4. (Syntax of $\mathcal{ALC}(\Sigma)$) (Baader & Sattler, 2003) The notion of a concrete domain $\mathcal{D}$ as presented in Definition 2.5.1 is extended with a set of aggregations $\mathsf{agg}(\mathcal{D})$, where each $\Gamma \in \mathsf{agg}(\mathcal{D})$ is associated with a partial function $\Gamma^{\mathcal{D}}$ from the set of multisets of $\Delta^{\mathcal{D}}$ to $\Delta^{\mathcal{D}}$. The concrete domain extended with aggregations is denoted as Σ. The set of concrete features is defined as follows:

$$F := f \mid f_1 ... f_n \mid f_1 ... f_n \Gamma(R \circ f)$$

- Each feature name (atomic feature) $f \in N_F$ is a concrete feature,
- a feature chain $f_1 ... f_n$ is a concrete feature,
- an aggregation feature $f_1 ... f_n \Gamma(R \circ f)$ is a concrete feature, where $f, f_1, ..., f_n$ are feature names, R is a role name, and $\Gamma \in \mathsf{agg}(\mathcal{D})$ is an aggregation function.

Then the syntax of $\mathcal{ALC}(\Sigma)$-concepts is obtained from $\mathcal{ALC}(\mathcal{D})$-concepts by allowing the additional use of concrete features in the concrete domain construct $P(f_1, ..., f_n)$.

At this point, the syntax of $\mathcal{ALC}(\Sigma)$ has been defined. The rest is to define the semantics of $\mathcal{ALC}(\Sigma)$ to support the new concrete feature, i.e., the aggregation feature.

Definition 2.5.5. (Semantics of $\mathcal{ALC}(\Sigma)$) (Baader & Sattler, 2003) An $\mathcal{ALC}(\Sigma)$-interpretation is extended from an $\mathcal{ALC}(\mathcal{D})$-interpretation to interprets the new aggregation features. Note that the interpretation is restricted to be finite. In order to define the semantics of aggregation features, the multiset $M_a^{R \circ f}$, which maps each elecment $z \in \Delta^\Sigma$ to the number of R-successors of a that have z as f-successors, is defined as follows:

$$M_a^{R \circ f}(z) := \#\{b \in \Delta^\mathcal{I} \mid (a, b) \in R^\mathcal{I} \text{ and } f^\mathcal{I}(b) = z\}$$

Now, the semantics of aggregation feature can be defined as follows:

$$(f_1 ... f_n \Gamma(R \circ f))^\mathcal{I}(a) := \begin{cases} \Gamma^\Sigma(M_{a'}^{(R \circ f)}), & \text{if } (f_1 ... f_n)^\mathcal{I}(a) = a' \wedge a' \in \Delta^\mathcal{I} \\ \text{undefined}, & (f_1 ... f_n)^\mathcal{I}(a) \notin \Delta^\mathcal{I} \end{cases}$$

and $\Gamma^\Sigma(M_{a'}^{(R \circ f)})$ is called the $(f_1 ... f_n \Gamma(R \circ f))$-successor of a.

There are two consequences of this definition that might not be obvious:

a An R-successor b of a with an abstract f-successor has no influence on $M_a^{R \circ f}$. $M_a^{R \circ f}$ is defined to handle $R \circ f$-successors of a in the concrete domain $\Delta^\mathcal{D}$.

b $M_{a'}^{(R \circ f)}$ needs to be a finite multiset because $\Delta^\mathcal{I}$ is finite. However, $(f_1 ... f_n \Gamma(R \circ f))^\mathcal{I}(a)$ might be undefined by two reasons: (1) there is no $f_1 ... f_n$-successors a' of a in $\Delta^\mathcal{I}$ and (2) aggregation functions can be partial. For example, the aggregation min over an empty set is undefined.

In the following, we will define the aggregation functions sum, count, max, and min using the multiset definition (Baader & Sattler, 2003). For finite multisets M over rational numbers, we can define the above aggregations as follows:

$$\mathsf{sum}(M) = \sum_{x \in M} M(x) \cdot x$$

$$\mathsf{count}(M) = \sum_{x \in M} M(x)$$

$$\mathsf{max}(M) = \begin{cases} m, & \text{if there exists } m \in M \text{ such that } n \leq m \text{ for all } n \in M \\ \text{undefined}, & \text{if no such } m \text{ exists} \end{cases}$$

$$\mathsf{min}(M) = \begin{cases} m, & \text{if there exists } m \in M \text{ such that } n \geq m \text{ for all } n \in M \\ \text{undefined}, & \text{if no such } m \text{ exists} \end{cases}$$

By now, the syntax and semantics of $\mathcal{ALC}(\Sigma)$ are defined. However, reasoning of $\mathcal{ALC}(\Sigma)$ is undecidable (Baader & Sattler, 2003). In order to retain decidability, three approaches were studied in (Baader & Sattler, 2003):

1. Decidability can be retained by disallowing universal restrictions of $\mathcal{ALC}(\Sigma)$, resulting in the Description Logic $\mathcal{EL}(\Sigma)$.

2. Decidability can be retained by disallowing the interaction between aggregation functions and value restrictions, i.e., existential and universal restrictions.

3. Decidability can be retained by restricting aggregation functions to contain only min and max.

In the following, we introduce $\mathcal{EL}(\Sigma)$ (Baader & Sattler, 2003) in detail since the logic considered in this thesis is based on this logic. $\mathcal{EL}(\Sigma)$ is obtained from $\mathcal{ALC}(\Sigma)$ by disallowing universal restrictions and restricting the use of negation to concept names. An $\mathcal{EL}(\Sigma)$-concept can be defined through the language constructs listed in the upper part of Table 2.7, and the lower part of Table 2.7 presents feature constructs. Note that the existential restriction is defined not only for roles, but also features.

Table 2.7: The syntax and semantics of $\mathcal{EL}(\Sigma)$.

Concepts	**Syntax**	**Semantics**
atomic concept	A	$A^{\mathcal{I}}$
atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conjunction	$\hat{C} \sqcap \hat{D}$	$\hat{C}^{\mathcal{I}} \cap \hat{D}^{\mathcal{I}}$
disjunction	$\hat{C} \sqcup \hat{D}$	$\hat{C}^{\mathcal{I}} \cup \hat{D}^{\mathcal{I}}$
existential restriction	$\exists R.\hat{C}$	$\{a \in \Delta^{\mathcal{I}} \mid$ there exists $b \in \Delta^{\mathcal{I}}$ such that $(a,b) \in R^{\mathcal{I}}$ and $b \in \hat{C}^{\mathcal{I}}\}$
existential feature restriction	$\exists f.\hat{C}$	$\{a \in \Delta^{\mathcal{I}} \mid$ there exists $b \in \Delta^{\mathcal{I}}$ such that $(a,b) \in f^{\mathcal{I}}$ and $b \in \hat{C}^{\mathcal{I}}\}$
concrete domain	$P(u_1, ..., u_n)$	$\{a \in \Delta^{\mathcal{I}} \mid \exists x_i$ such that $u_i^{\mathcal{I}}(a) = x_i$ and $(x_i, ..., x_n) \in P^{\Sigma}\}$
Features	**Syntax**	**Semantics**
atomic feature	f	$\Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}} \cup \Delta^{\Sigma}$
feature chain	$f_1 ... f_n$	$f_n^{\mathcal{I}}(f_{n-1}^{\mathcal{I}}(...(f_1^{\mathcal{I}}(a)...) \wedge a \in \Delta^{\mathcal{I}}$
aggregation	$\Gamma(R \circ f)$	$\begin{cases} \Gamma^{\Sigma}(M_a^{(R\circ f)^I}), & \text{if } M_a^{(R\circ f)^I} \text{is a multiset} \\ \text{undefined}, & \text{otherwise} \end{cases}$

The reasoning service considered for $\mathcal{EL}(\Sigma)$ is *concept satisfiability* without a TBox. In general, the decidability of concept satisfiability of $\mathcal{EL}(\Sigma)$ depends on the decidability of concrete domains Σ. Concept satisfiability of $\mathcal{EL}(\Sigma)$ is decidable if and only if the consistency of concrete domains Σ (Σ-*consistency*) is decidable. Concept satisfiability of $\mathcal{EL}(\Sigma)$ was proven decidable, where Δ^{Σ} is the set of non-negative integers, integers, or rational numbers and all of them involving only the aggregation functions min, max, and count.

Using aggregations as concept constructs is the main focus of this thesis. There are a number of related studies about aggregations in conjunctive query answering:

1. Aggregations were investigated for query answering for the ontology language $DL-Lite_{\mathcal{A}}$ (Calvanese, Kharlamov, Nutt, & Thorne, 2008). The syntax and semantics were proposed for (conditional) epistemic aggregate queries over ontologies and query answering for max, min, count, cntd (count distinct), sum, and avg (average) was studied. This allows us to use aggregation query over ontologies.

2. Some research focused on the aggregation count. The semantics for answering counting aggregate queries, which are queries that use count and cntd (count distinct) were studied further (Kostylev & Reutter, 2013). The aim of this approach is to find the

maximal information in the answers of these queries for all the models of a knowledge base. This research concentrated on query answering for ontology languages $DL - Lite_{core}$ and $DL - Lite_{\mathcal{R}}$ (Kostylev & Reutter, 2013). In addition, the complexity of counting aggregate queries for such ontology languages was studied. It turned out that the complexity is coNP-complete. The need and relationship of different interpretation of count distinct aggregation were studied (Kostov & Křemen, 2013). There are four different interpretations, which are basic count, semantic count, epistemic count, semantic tuple count, were studied.

3. Aggregations were studied for real-time analytics that need to aggregates distributed streaming and static data (Kharlamov et al., 2016). In particular, this research aimed at supporting common analytical tasks for Siemens turbine diagnostics, which need to deal with streaming data produced by up to 2,000 sensors. The ontology language $DL - Lite_{\mathcal{A}}^{\mathsf{agg}}$ (Kharlamov et al., 2016), which extends $DL - Lite_{\mathcal{A}}$ with aggregations as first class citizen (concepts that are based on aggregation of attribute values), was proposed.

As can be seen, most of recent studies mainly focus on aggregations in conjunctive query answering. It is interesting to revisit and investigate the aggregations as concept construct further since the ability to express aggregation over concrete domains is useful in many domains and situations such as fitness tracking application as shown in Figure 2.2.

2.6 Relationship to OWL

Ontology has different meanings in different areas. For example, generally, ontology is a vocabulary which can represent conceptualisation in particular domain (Chandrasekaran, Josephson, & Benjamins, 1999). In the perspective of Semantic Web, ontologies are building blocks of knowledge (Heflin, 2004) and it is used to help automated processes to access information. For example, ontologies could be used to facilitate negotiation between buying and selling agents in e-commerce (Tamma, Phelps, Dickinson, & Wooldridge, 2005) or it can be used to find pages that contain semantically similar words and phrases (Lei, Uren, & Motta, 2006). In general, ontologies are used to represent *classes*, *relationship between classes*, and *properties*, that those classes may have. Importantly, ontologies needs to have appropriate structure. As a result, W3C Recommendation recommends Web Ontology Language (OWL) for building ontologies in Semantic Web.

OWL 1 was introduced as a new language for modelling ontologies in Semantic Web (Horrocks et al., 2003). The Description Logic $\mathcal{SHOIN}$ with data type feature ($\mathbf{D}$), which is denoted as $\mathcal{SHOIN}(\mathbf{D})$ (Horrocks & Patel-Schneider, 2004), provides the underpinning semantics for OWL 1 (Grau et al., 2008). In addition, as RDF extension, OWL 1 has fact-stating feature of RDF (Cyganiak, Wood, & Lanthaler, 2014) and the class- and property-structuring capability of RDF Schema (RDFS). For example, OWL 1 can declare classes and supports subsumption of classes, and it can declare properties and supports subproperty of properties (Horrocks et al., 2003). In addition, OWL 1 extends the capability of RDFS in crucial ways. For example, OWL 1 supports class declaration as logical combinations such as intersections, unions, and complements (Horrocks et al., 2003). OWL 1 also supports characteristics of a property such as transitive and functional.

There are three sublanguages of OWL 1, which are *OWL 1 Full*, *OWL 1 DL*, and *OWL 1 Lite* (McGuinness & van Harmelen, 2004). Furthermore, each sublanguage is developed for specific applications. OWL Full fully supports capability of RDF and RDFS and all RDF graphs are allowed. OWL 1 Full is undecidable and difficult to implement (Grau et al., 2008). On the other hand, OWL DL only allows certain language constructs and combinations of language constructs are limited. OWL 1 DL is developed as variant of the Description Logic $\mathcal{SHOIN}(\mathbf{D})$. Therefore, the syntax of OWL 1 DL is very close to that of $\mathcal{SHOIN}(\mathbf{D})$. OWL 1 DL is for applications that consider decidable inference and friendly syntax as importances. However, OWL 1 DL is sometimes difficult for naive users and $\mathcal{SHOIN}(\mathbf{D})$ is difficult to reason about as its complexity is NExpTime-complete. As as result, OWL 1 Lite is proposed as a syntactic subset of OWL 1 DL. OWL Lite disallows unions, complements, individuals to show up in descriptions or class axioms. In addition, it restricts intersections to the implicit intersections in the frame-like class axioms and cardinalities to 0 or 1. Due to these restrictions, OWL 1 Lite is very close to the Description Logic $\mathcal{SHIF}(\mathbf{D})$ with the complexity of ExpTime-complete (Tobies, 2001). In addition, there are practical optimised algorithms for reasoning about OWL 1 Lite such as the algorithm underlying the Description Logic system FaCT (Horrocks, 1998).

OWL 1 has many limitations, such as expressivity limitations and syntax issues, as summarised in (Grau et al., 2008). It has several expressivity limitations such as the lack of qualified cardinality restrictions, weak relationship expressivity and datatype expressivity limitations. For example, considering qualified cardinality restrictions, OWL 1 does not allow user to express *a cat with at least four children which are kittens.* Additionally, OWL 1 has several syntax issues. OWL 1 comes with two normative syntaxes: (1) the Abstract Syntax (Patel-Schneider, Hayes, & Horrocks, 2004) and (2) OWL 1 RDF (Bechhofer et al., 2004). The Abstract Syntax is inspired by the tradition of frame-based ontology languages. With this syntax, the relationship between class declaration and axiom declaration is a major source of confusion (Grau et al., 2008). Another issues is alignment with DL constructs. Although OWL 1 is based on DLs, the constructs of OWL 1 is not related to the constructs of DLs. For OWL 1 RDF syntax, it is difficult to represent relationship of object in RDF without introducing new objects (Grau et al., 2008). Due to this reason, OWL 1 syntaxes are difficult to read and not practical. In addition, the scalability of OWL 1 systems is limited.

Due to limitations of OWL 1, OWL 2 is introduced as the extension of OWL 1 with more expressive power (Grau et al., 2008). Likewise, OWL 2 is based on the most expressive DL, $\mathcal{SROIQ}(\mathcal{D})$ (Horrocks et al., 2006). In addition, OWL 2 use Meta-Object Facility (MOF) metamodel to specify its structure to overcome OWL 1 syntax issues. Accordingly, this helps OWL 2 to have more readable syntax than that of OWL 1. In addition to MOF, *Functional-Style Syntax* is introduced to be a simple syntax for OWL 2. Although, the basic DL of OWL 2 is the decidable $\mathcal{SROIQ}(\mathcal{D})$, reasoning complexity of OWL 2 is high, N2ExpTime-complete. As a result, the OWL Working group considers that it is crucial to define fragments of OWL 2, which have lower complexity and are easier to use. For this purpose, the OWL Working group defines three profiles of OWL 2, which are *OWL 2 EL*, *OWL 2 QL* and *OWL 2 RL* (Motik et al., 2012). Accordingly, each profile provides different expressive power and focuses on different applications. Table 2.8 shows a summary of specification and complexity of main reasoning tasks of OWL 2 profiles and OWL 2 DL. The complexity here is the combination between taxonomic complexity, data complexity, and query complexity. The reasoning services considered are consistency checking, concept satisfiability, concept subsumption, and instance checking.

There are two styles of formal semantics of OWL: (1) *Direct Semantics* (Horrocks, Parsia, & Sattler, 2012) based on DLs and (2) *RDF-Based Semantics* (Carroll, Herman, & Patel-Schneider, 2012). The Direct Semantics of OWL is only defined for a certain fragment of OWL, which is called OWL DL (or OWL 2 DL). For OWL Full, it can be interpreted under the RDF-Based Semantics.

The OWL standard provides a number of syntactic forms that can be used to express OWL ontologies such as RDF/XML, Manchester, and Functional-style syntax. The most

Table 2.8: Summary of specification and complexity of main reasoning tasks of OWL 2 profile (Motik et al., 2012).

Profile	**Description Logic**	**Description**	**Complexity**
OWL 2 DL	$\mathcal{SROIQ}(\mathcal{D})$ (Horrocks et al., 2006)	The most expressive OWL 2 profile.	N2ExpTime-complete
OWL 2 EL	$\mathcal{EL}^{++}$(Baader, Brandt, & Lutz, 2005)	This profile is designed for large ontologies, which have large number of classes and have tractable reasoning complexity.	PTime-complete
OWL 2 QL	$\mathit{DL}\text{-}\mathit{Lite}^{\mathcal{H}}_{\mathit{core}}$ (Artale, Calvanese, Kontchakov, & Zakharyaschev, 2009)	This profile is designed to support applications, which have large amount of data.	NLogSpace-complete
OWL 2 RL	*DLP* (Grosof, Horrocks, Volz, & Decker, 2003)	This profile is designed for applications, which need scalable rule-based reasoning without losing expressive power.	coNP-complete

important one is RDF/XML since this syntax style is well-known and most OWL tools support. In contrast, the Functional-style syntax is the most readable.

Chapter 3

Reasoning Algorithms for Description Logics

The reasoning services that this thesis focus on are terminological reasoning (TBox reasoning) including *concept satisfiability*, *concept subsumption*, and *consistency checking.* These reasoning services are described in Section 2.3.

Over the years, in order to conduct reasoning for Description Logics, the algorithm called *Structural Subsumption Algorithm* was used in the early days (Baader & Nutt, 2003). This algorithm compares the syntactic structure of concept in order to check concept subsumption of some Description Logics. However, these algorithms are not complete for Description Logics with (full) negation and disjunction. At the present time, two broad types of the-state-of-the-art algorithms have been proposed. The first group of algorithms is called *Tableau-based Algorithms* (Schmidt-Schauß & Smolka, 1991), which are based on tableau calculi, and can overcome the incompleteness of the structural subsumption algorithms (Baader & Nutt, 2003). In addition, these algorithms have turned out to be useful for solving concept satisfiability and concept subsumption (Schmidt-Schauß & Smolka, 1991; Hollunder & Baader, 1991; Baader, 1991; Hollunder, Nutt, & Schmidt-Schauß, 1990) over highly expressive Description Logics. For light weight Description Logics such as $\mathcal{EL}$ (Baader, 2003), the tableau-based algorithms are not optimal. Thus, the second group of algorithms called *Polynomial-Time Subsumption (Classification) Algorithms* (Brandt, 2004; Baader, Lutz, & Suntisrivaraporn, 2005) have been proposed. These algorithms can simultaneously compute the subsumption relationships between all pairs of concept names.

In this chapter, we provide an overview of the tableau-based algorithms and polynomial-time subsumption (classification) algorithms. In addition, some examples of both algorithms are provided.

3.1 Tableau-based Algorithms

The first tableau-based algorithm was introduced in (Schmidt-Schauß & Smolka, 1991) as a solution to the concept satisfiability problem of the description logic $\mathcal{ALC}$. Given a concept $\hat{C}$, the tableau-based algorithms test the satisfiability of the concept $\hat{C}$ by attempting to construct a model of $\hat{C}$ as a tree structure where the branches are closed by conflicts. We use the constraint notation (Hollunder, 1990) to describe the tableau algorithm. A model of $\hat{C}$ is represented as a *completion constraint system* $\mathcal{A}$. Using the constraint notation, individuals in $\mathcal{A}$ are represented by lower case letters. For each individual a, if there is an axiom $a : C$ of a concept C in $\mathcal{A}$, it means that the individual corresponding with a is in the interpretation of C in the model. For each pair (a, b), if there is an axiom $(a, b) : R$ of a role name R in $\mathcal{A}$, it means that the pair corresponding with (a, b) is in the interpretation of R in the model. A letter b is called *R-successor* of a if there exists an axiom $(a, b) : R$ in $\mathcal{A}$.

The tableau algorithm starts from the root individual a_0 such that $\mathcal{A} = \{a_0 : \hat{C}\}$. The algorithm terminates either when the constraint system is *complete* (no expansion rules can be applied further), or when an obvious contradiction (or a *clash*), i.e., $\{a : C, a : \neg C\} \subseteq \mathcal{A}$), is detected in all branches. The tableau algorithm transforms an input concept to *Negation Normal Form* (NNF) (see Definition 2.4.1). For example, $\neg\exists R.(A \sqcap \neg B)$ can be transformed to $\forall R.(\neg A \sqcup B)$, where A and B are atomic concept and R is a role name. After that, the tableau algorithm applies expansion rules to construct a model of the concept. The expansion rules are rules used to expand the search tree to find a contradiction. The expansion rules can be found in Figure 3.1. (see (Baader & Sattler, 2001) for details). If there is no contradiction, the concept is satisfiable. The expansion rules can be deterministic and non-deterministic. Since the disjunction is handled by non-deterministic rules, they can cause branching during the reasoning process exponentially and thus decrease the performance of this type of algorithms. The application of non-deterministic rules can be considered as OR-search which is one of the major sources of inefficiency of tableau-based algorithms (Motik et al., 2007b, 2007a, 2009).

$$
\begin{aligned}
\sqcap\text{-rule} :& \text{if } 1. a : (C_1 \sqcap C_2) \in \mathcal{A}, \text{ and} \\
& \qquad 2. \{a : C_1, a : C_2\} \not\subseteq \mathcal{A}, \\
& \text{then } \mathcal{A}' := \mathcal{A} \cup \{a : C_1, a : C_2\} \\
\sqcup\text{-rule} :& \text{if } 1. a : (C_1 \sqcup C_2) \in \mathcal{A}, \text{ and} \\
& \qquad 2. \{a : C_1, a : C_2\} \cap \mathcal{A} = \emptyset, \\
& \text{then } \mathcal{A}' := \mathcal{A} \cup \{a : C_1\} \text{ or } \mathcal{A}'' := \mathcal{A} \cup \{a : C_2\} \\
\exists\text{-rule} :& \text{if } 1. a : (\exists R.C) \in \mathcal{A}, \text{ and} \\
& \qquad 2. \text{there is no } b \text{ such that } (a, b) : R \in \mathcal{A} \text{ and } b : C \in \mathcal{A}, \\
& \text{then create a new individual } b \text{ with } \mathcal{A}' := \mathcal{A} \cup \{(a, b) : R, b : C\} \\
\forall\text{-rule} :& \text{if } 1. a : (\forall R.C) \in \mathcal{A}, \text{ and} \\
& \qquad 2. \text{there is some } b \text{ such that } (a, b) : R \in \mathcal{A} \text{ and } b : C \notin \mathcal{A}, \\
& \text{then } \mathcal{A}' := \mathcal{A} \cup \{b : C\}
\end{aligned}
$$

Figure 3.1: Tableau expansion rules for $\mathcal{ALC}$.

Now, we will introduce each tableau expansion rule. The conjunction rule ($\sqcap$-rule) is applied to an individual a, where it is a member of a concept definition $C_1 \sqcap C_2$, which is a conjunction. However, the rule is not applied to an individual a if a is already asserted to be a member of all of the conjuncts of $C_1 \sqcap C_2$ (i.e., $\{a : C_1, a : C_2\} \subseteq \mathcal{A}$). The result of the conjunction rule is the individual a being asserted as a member of each conjunct (e.g. $\mathcal{A} \cup \{a : C_1, a : C_2\}$). The disjunction rule ($\sqcup$-rule) is applied to an individual a, where it is a member of a concept definition $C_1 \sqcup C_2$, which is a disjunction. However, the rule is not applied to an individual a if a is already asserted to be a member of any of the disjuncts of $C_1 \sqcup C_2$ (i.e., $\{a : C_1, a : C_2\} \cap \mathcal{A} \neq \emptyset$). The result of the disjunction rule is the individual a being asserted as a member of one of the disjuncts (e.g. $\mathcal{A} \cup \{a : C_1\}$ or $\mathcal{A} \cup \{a : C_2\}$). The existential restriction rule ($\exists$-rule) is applied to an individual a, where it is a member of a concept definition $\exists R.C$, which is an existential restriction. However, the rule is not applied to an individual a if a has an R-successor b, where b is asserted as a member of the concept C (i.e., $(a, b) : R \in \mathcal{A}$ and $b : C \in \mathcal{A}$). The result of the existential restriction expansion rule is an individual b is created as an R-successor of the individual a and the individual b being asserted as a member of the concept C (e.g. $\mathcal{A} \cup \{(a,b) : R, b : C\}$). The universal restriction rule ($\forall$-rule) is applied to an individual a, where it is a member of a concept definition $\forall R.C$, which is an universal restriction. However, the rule is only applied to an individual a, when a has at least one R-successor b, where b is not asserted as a member of the concept name C and b is connected to the individual a via the role R (i.e., $(a,b) : R \in \mathcal{A}$ and $b : C \notin \mathcal{A}$). The result of the universal restriction expansion rule is all R-successors b of the individual a being asserted as members if the concept name C (e.g. $\mathcal{A} \cup \{b : C\}$).

These algorithms then have been extended to support sound and complete reasoning for concept satisfiability checking on the general TBox (Baader & Sattler, 2001; Baader & Nutt, 2003), and to handle consistency checking of ABoxes (Baader, Horrocks, & Sattler, 2008) respectively. The tableau algorithms are widely used for highly expressive DLs such as $\mathcal{ALC}$ with extensions such as qualified number restriction and concrete domains (Baader & Sattler, 2001).

Over the years, a number of optimisation techniques have been proposed. The application of expansion rules on qualified number restriction can increase the search space exponentially (Baader, Horrocks, & Sattler, 2008). For some highly expressive Description Logic with general TBox, a technique called *blocking* (Buchheit, Donini, & Schaerf, 1993) is used to ensure that the tableau algorithm terminates. These algorithms have been implemented in many state-of-the-art reasoners such as FaCT++ (Tsarkov & Horrocks, 2006), Pellet (Sirin et al., 2007), and Konclude (Steigmiller et al., 2014). Absorption, which is a rewriting technique used to reduce the number of GCIs, have been developed to improve the performance of these algorithms.

Several researchers also introduced a novel reasoning algorithm called *Hyper-tableau Algorithm* (Motik et al., 2007b, 2007a, 2009). This algorithm is based on hyperresolution with anywhere blocking. This algorithm has been implemented in an ontology reasoner called HermiT (Shearer et al., 2008; Glimm et al., 2014).

Nonetheless, these algorithms still have difficulties in handling highly expressive DLs and the inefficiencies of this type of algorithms still remain when they are used to perform reasoning on large and complex ontologies, which contain many disjunctions, GCIs, qualified number restrictions, and concrete domains, such as Basic Call System (BCS) (Areces, Bouma, & de Rijke, 1999) and the Genomic CDS ontology (Kang et al., 2012, 2014, 2015).

3.1.1 Tableau-based Algorithms for Concrete Domain

In order to handle concrete domain constructs ($P(u_1, ..., u_n)$) of Description Logic $\mathcal{ALC}(\mathcal{D})$ that we described in Section 2.5.1, One more rule (shown in Figure 3.2) was added in (Baader & Hanschke, 1991). Note that we call this rule is called P-rule in this thesis. Let us describe how to extend the constraint system explained above. We need to recall some notations. Let f represent a feature name, $a, b, c, ..$ represent abstract individuals in $\Delta^{\mathcal{I}}$, $x, y, ...$ represent concrete individuals in $\Delta^{\mathcal{D}}$.

For feature chains $u_i = f_{i1}...f_{im_i}$, we use an axiom $(a,b) : f$ or $(a,x) : f$ for each f, where $b \in \Delta^{\mathcal{I}}$ and $x \in \Delta^{\mathcal{D}}$. Either (a,b) or (a,x) is in the interpretation of f. Individuals b and x are f-successors of a if there exists axioms $(a,b) : f$ or $(a,x) : f$ in $\mathcal{A}$ respectively. An axiom $(x_1, ..., x_n) : P$ means that the tuple corresponding with $(x_1, ..., x_n)$ is in the interpretation of a predicate P. If $\mathcal{A}$ contains a pair of axioms $(a,b) : f$ and $(a,c) : f$, or $(a,x) : f$ and $(a,y) : f$, such a pair of axioms is called a *fork* in $\mathcal{A}$. Such a fork means that b and c, or x and y have to be interpreted as the same individual since f is interpreted as a partial function. Therefore, a fork $(a,b) : f$ and $(a,c) : f$, or $(a,x) : f$ and $(a,y) : f$ can be deleted by replacing all occurrences of c in $\mathcal{A}$ by b, or replacing all occurrences of y in $\mathcal{A}$ by x. After applying the P-rule, a constraint system $\mathcal{A}$ is generated with the corresponding conjunction $c_{\mathcal{A}} = \bigwedge_{i<k} P_i(x_1^{(i)}, ..., x_{n_i}^{(i)})$. Note that since existential

and universal restrictions can be used with a feature name, the $\exists$ and $\forall$ rules in Figure 3.1 can be applied where a feature name f is considered instead of a role name R.

P-rule :if 1.$a : P(u_1, ..., u_n) \in \mathcal{A}$, and

2. for feature chain $u_i = f_{i1}...f_{im_i}$, there is no $b_{i1}, ...b_{im_i-1} \in \Delta^{\mathcal{I}}$ and $x_i \in \Delta^{\mathcal{D}}$ such that $\{(a, b_{i1}) : f_{i1}, ..., (b_{im_i-1}, x_i) : f_{im_i}\} \not\subset \mathcal{A}$ and $(x_1, ..., x_n) : P \notin \mathcal{A}$,

then for the feature chain $u_i = f_{i1}...f_{im_i}$,

creating new individuals $b_{i1}, ...b_{im_i-1} \in \Delta^{\mathcal{I}}$ and $x_i \in \Delta^{\mathcal{D}}$,

$\mathcal{A}' := \mathcal{A} \cup \{(a, b_{i1}) : f_{i1}, ..., (b_{im_i-1}, x_i) : f_{im_i}\}$

Finally, $\mathcal{A}' := \mathcal{A} \cup \{(x_1, ..., x_n) : P\}$

Figure 3.2: Tableau expansion rules for concrete domains.

The constraint system $\mathcal{A}$ for $\mathcal{ALC}(\mathcal{D})$ is *satisfiable* if and only if $c_{\mathcal{A}}$ is consistent ($\mathcal{D}$-consistency). The constraint system $\mathcal{A}$ for $\mathcal{ALC}(\mathcal{D})$ contains a *clash* if and only if one of the following situations occurs in $\mathcal{A}$:

1. $\{(a, b) : f, (a, x) : f\} \subseteq \mathcal{A}$. This is an obvious contradiction since the concrete individual should not be related to the same feature name as the abstract individual.

2. $\{(a, x) : f, a : \forall f.C\} \subseteq \mathcal{A}$. This is an obvious contradiction since a concrete individual cannot be an element of a concept

3. $\{a : C, a : \neg C\} \subseteq \mathcal{A}$. This is an obvious contradiction since an individual cannot be in both a concept and its complement.

4. $\{(x_1^{(1)}, ..., x_{n_1}^{(1)}) : P_1, ..., (x_1^{(k)}, ..., x_{n_k}^{(k)}) : P_k\} \subseteq \mathcal{A}$ and the corresponding conjunction $\bigwedge_{i<k} P_i(x_1^{(i)}, ..., x_{n_i}^{(i)})$ is not satisfiable in $\mathcal{D}$.

If $\mathcal{A}$ does not contain a clash, $\mathcal{A}$ is said to be *clash-free*.

Let us use Example 3.1.1 to illustrate the tableau algorithm for concrete domains.

Example 3.1.1. Let concept description C be defined as follow:

$$\begin{aligned} C := &\ \mathsf{Treadmill} \sqcap \exists \mathsf{runningsurface.RunningSurface} \sqcap \\ &=_{120} .(\mathsf{powerrequired}) \sqcap\ > .(\mathsf{width}, (\mathsf{runningsurface\ width})) \end{aligned}$$

After applying the tableau expansion rules in Figure 3.1, we obtain the following constraint system $\mathcal{A}$:

$$\begin{aligned} \mathcal{A} = \{ & a : \mathsf{Treadmill}, a : \exists \mathsf{runningsurface.RunningSurface}, (a, b) : \mathsf{runningsurface}, \\ & b : \mathsf{RunningSurface}, a :=_{120} .(\mathsf{powerrequired}), a :> .(\mathsf{width}, (\mathsf{runningsurface\ width}))\} \end{aligned}$$

Considering $a :=_{120} .(\mathsf{powerrequired})$, If we apply the P-rule to this assertion, the P-rule will introduce $\mathsf{powerrequired}$-successor x. Then assertions $(a, x) : \mathsf{powerrequired}$ and $(x) :=_{120}$ are added to $\mathcal{A}$ and the conjunction of $\mathcal{D}$-consistency $\mathcal{A}_{\mathcal{D}}$ is generated with the first constraint $=_{120} (x)$ as shown below:

$$\begin{aligned} \mathcal{A}' &= \mathcal{A} \cup \{(a, x) : \mathsf{powerrequired}, (x) :=_{120}\} \\ \mathcal{A}_{\mathcal{D}} &= (=_{120} (x)) \end{aligned}$$

Considering $a :> .(\mathsf{width}, (\mathsf{runningsurface\ width}))$, If we apply the P-rule to this assertion, the P-rule will introduce width-successor y and (runningsurface width)-successor z. Then assertions $(a, y) : \mathsf{width}$, $(a, z) : (\mathsf{runningsurface\ width})$, and $(y, z) :>$ are added to $\mathcal{A}'$ and the constraint $> (y, z)$ is added to $\mathcal{A}_\mathcal{D}$ as shown below:

$$\mathcal{A}'' = \mathcal{A}' \cup \{(a, y) : \mathsf{width}, (a, z) : (\mathsf{runningsurface\ width}), (y, z) :>\}$$
$$\mathcal{A}'_\mathcal{D} = (=_{120} (x)) \wedge (> (y, z))$$

Since there is no clash in $\mathcal{A}''$ and $\mathcal{A}'_\mathcal{D}$ is consistent, the concept C is satisfiable. □

3.1.2 Tableau-based Algorithms for Aggregations

Moreover, the tableau algorithm was extended to support aggregation mentioned in Section 2.5.2 (Baader & Sattler, 2003) for $\mathcal{EL}(\Sigma)$. The main idea of this tableau algorithm is to check a model for the abstract domain using expansion rules and then check the consistency of the conjunction of constraints of the concrete domain. Now, let us describe the tableau algorithm for the Description Logic $\mathcal{EL}(\Sigma)$ in Section 2.5.2. The tableau expansion rule for concrete domain P-rule in Figure 3.2 and $\exists$-rule in Figure 3.1 need to be modified, as shown in Figure 3.3. The modified rules are referred as MP-rule and $M\exists$-rule respectively. The modified rule MP-rule is used to handle concrete domain and aggregations $\Gamma(R \circ f)$. In addition to the rule modifications, one more rule, called E-rule, needs to be added to assert elements in a multiset. This rule is used to ensure that if a R-successor b of a has f-successor z and $(a, X) : (R \circ f) \in \mathcal{A}$, the f-successor z needs to be in the multiset X. Before we describe the new expansion rules, let us give some technical definitions. Let $\Delta^\mathcal{I}$ represent a set of abstract individuals, $\Delta^\mathcal{D}$ represent a set of concrete individuals, $\mathsf{agg}(\mathcal{D})$ be a set of aggregation functions, $\mathsf{agg}(\mathcal{MS})$ represent the set of aggregated variables, $\{\Gamma(X) \mid \Gamma \in \mathsf{agg}(\mathcal{D})$ and $X \in \mathcal{MS}\}$, Δ^Σ denote $\Delta^\mathcal{D}$ that provides aggregations, and $\mathcal{MS} = \{X, Y, Z, ...\}$ represents a set of multiset variables. Three new types of constraint axioms were introduced to handle aggregation as follows:

1. $(a, Y) : (R \circ f)$ for $a \in \Delta^\mathcal{I}$, $R \in N_R$, $f \in N_F$, $Y \in \mathcal{MS}$,
2. $(\alpha_1, ..., \alpha_n) : P$ for $\alpha_i \in \Delta^\Sigma$, and
3. $x : Y$ for $x \in \Delta^\mathcal{D}, Y \in \mathcal{MS}$

An aggregated variable $\Gamma(X)$ is said to be an $f_1...f_n\Gamma(R \circ f)$-successor of a in $\mathcal{A}$ if and only if there exists an $f_1...f_n$-successor b of a in $\mathcal{A}$ and $(b, X) : (R \circ f) \in \mathcal{A}$. The constraint axiom $(\alpha_1, ..., \alpha_n) : P$ is the same as the constraint axiom $(x_1, ..., x_n) : P$, except that the domain of α is Δ^Σ but the domain of x is $\Delta^\mathcal{D}$. The constraint axiom $x : Y$ means that the individual x is in the interpretation of the multiset Y. α_i can be either a concrete individual or an aggregated variable $\Gamma(X)$. If $\alpha_i = \Gamma(X)$, then α_i is the result of aggregation Γ over multiset X. If $\mathcal{A}$ contains the axioms $(a, X) : (R \circ f)$ and $(a, Y) : (R \circ f)$, such a pair of axioms is called a *fork* in $\mathcal{A}$. This fork can be eliminated by replacing each occurrence of X by Y. The constraint system $\mathcal{A}$ is satisfiable if and only if the consistency of concrete domains Σ (Σ-*consistency*) is consistent. Σ-consistency consists of two types of Σ-constraints $(\alpha_1, ..., \alpha_n) : P$ and $x : Y$. In order to check Σ-consistency, the conjunction $\mathcal{A}_\Sigma$ is generated as follow:

$$\mathcal{A}_\Sigma := \bigwedge_{(\alpha_1,...,\alpha_n):P \in \mathcal{A}} P(\alpha_1, ..., \alpha_n) \wedge \bigwedge_{Y \text{ occurs in } \mathcal{A}} [\![x_i \mid x_i : Y \in \mathcal{A}]\!] \subseteq Y$$

$M\exists$-rule :if $1.a : (\exists R.C) \in \mathcal{A}$
 2. for a role name $R, \{b_1, ..., b_n\}$ are all R-successors of a and $b_i : C \notin \mathcal{A}$ for all b_i
 then $\mathcal{A}_i := \mathcal{A} \cup \{b_i : C\}$
 $\mathcal{A}_{n+1} := \mathcal{A} \cup \{(a, b) : R, b : C\}$ for a new individual b

MP-rule :if $1.a : P(u_1, ..., u_n) \in \mathcal{A},$ and
 2. for the feature chain u_i, there is no u_i-successor α_i of a and
 $(\alpha_1, ..., \alpha_n) : P \notin \mathcal{A},$
 then for each feature chain u_i,

$$\mathcal{A}_i := \begin{cases} \{(a, b_{i1}) : f_{i1}, ..., (b_{im_i - 1}, x_i) : f_{im_i}\}, \\ \qquad \text{if } u_i = f_{i1}...f_{im_i} \\ \{(a, b_{i1}) : f_{i1}, ..., (b_{im-1}, b_{im_i}) : f_{im_i}, (b_{im_i}, X_i) : (R_i \circ f_i)\}, \\ \qquad \text{if } u_i = f_{i1}...f_{im_i}\Gamma_i(R_i \circ f_i) \end{cases}$$

 for new individuals $b_{ij} \in \Delta^{\mathcal{I}}, x_i \in \Delta^{\Sigma}$ and multiset variable $X_i \in \mathcal{MS}$.
 Let α_i be the u_i-successor of a in $\mathcal{A}_i$, where α_i can be either x_i or $\Gamma(X_i)$
 Finally, $\mathcal{A}' := \mathcal{A} \cup \{(\alpha_1, ..., \alpha_n) : P\} \cup \bigcup_{1 \le i \le n} \mathcal{A}_i$

E-rule :if $1.\{(a, b) : R, (b, z) : f, (a, X) : (R \circ f)\} \subseteq \mathcal{A},$ and
 $2.z : X \notin \mathcal{A}$
 then $\mathcal{A}' := \mathcal{A} \cup \{z : X\}$

Figure 3.3: Tableau expansion rules for aggregations.

The Σ-consistency constraint $\mathcal{A}_\Sigma$ is consistent if and only if $\mathcal{A}_\Sigma$ has a solution. In addition to the fork condition previously mentioned, a constraint system $\mathcal{A}$ contains a fork if and only if $\{(a, X) : (R \circ f), (a, Y) : (R \circ f)\} \in \mathcal{A}$ for two distinct multisets $X, Y \in \mathcal{MS}$. The fork can be eliminated by replacing all occurrences of X by Y.

The $\exists$-rule need to be modified since the original $\exists$-rule in Figure 3.1 would cause incomplete constraint systems. Let us use Example 3.1.2 to illustrate this problem.

Example 3.1.2. Considering the aggregation count, given a concept C as follow:

$$C := (\exists R.P_{\geq 2}(f)) \sqcap (\exists R.P_{=2}(f)) \sqcap P_{\leq 1}(\mathsf{count}(R \circ f))$$

Applying the original $\exists$-rule in Figure 3.1, for an individual $a : C$, there are two R-successors, b, c generated for $\exists R.P_{\geq 2}(f)$ and $\exists R.P_{=2}(f)$ respectively. Then two $R \circ f$-successors, x, y, are generated for b and c respectively and added to a multiset Y after applying the MP-rule and E-rule. The Σ-consistency cannot be established since the conjunction of Σ-consistency constraints, which has $[\![x, y]\!] \subseteq Y \wedge (\mathsf{count}(Y) \leq 1)$, is inconsistent. This is because the number of elements in Y is actually 2, which contradicts the constraint $\mathsf{count}(Y) \leq 1$. In order to take this into account, the modified $\exists$-rule, $M\exists$-rule in Figure 3.3 tries to reuse all R-successors generated before generating a new one. Applying the $M\exists$-rule, for an individual $a : C$, the algorithm first generates one new R-successor, b, for $\exists R.P_{\geq 2}(f)$, then one $R \circ f$-successor, x, with f-value more than or equal to 2 for b. Next, the algorithm tries to reuse b with x for $\exists R.P_{=2}(f)$ instead of generating a new R-successor. Since the f-value of x is more than or equal to 2, it satisfies both $P_{\geq 2}(f)$ and $P_{=2}(f)$. As a result, the conjunction of Σ-consistency constraints has

$[\![x]\!] \subseteq Y \wedge (\mathsf{count}(Y) \leq 1)$ and it is consistent. However, if this reuse is not possible and x does not satisfy $P_{=2}(f)$, the algorithm will try to introduce a new R-successor for $\exists R.P_{=2}(f)$. This makes the $M\exists$-rule *non-deterministic.* □

3.2 Polynomial-Time Subsumption (Classification) Algorithms

For light weight Description Logic such as $\mathcal{EL}$, a polynomial-time, completion-based subsumption algorithm has been proposed (Brandt, 2004). In addition, this algorithm has been extended to support $\mathcal{EL}^{+}$ and $\mathcal{EL}^{++}$ (Baader, Lutz, & Suntisrivaraporn, 2005). This section will give an overview of the polynomial-time subsumption algorithm for $\mathcal{EL}$.

The Description Logic $\mathcal{EL}$ does not allow negation. Therefore, concept subsumption checking becomes the main problem of the $\mathcal{EL}$ family. The algorithm will perform the following four steps in order to check subsumption relationships of all concepts in a given TBox (Turhan, 2010).

1. Normalise a given TBox
2. Map the normalised TBox to completion sets
3. Apply completion rules to the completion sets
4. Read subsumption relationships in the completion sets

3.2.1 Normalisation

Given a TBox $\mathcal{T}$, the purpose of the algorithm is to find subsumption relationships of concepts and roles in $\mathcal{T}$ and to classify $\mathcal{T}$ (Baader, Brandt, & Lutz, 2005). Firstly, the algorithm normalises $\mathcal{T}$ to a normalised TBox $\mathcal{T}'$ by creating new concept names according to normalisation rules as described in Section 2.4.1 in linear time. Let us recall the normalisation rule as shown below:

$$\hat{C} \equiv \hat{D} \rightsquigarrow \hat{C} \sqsubseteq \hat{D}, \hat{D} \sqsubseteq \hat{C} \tag{NR1}$$
$$\hat{C} \sqsubseteq \hat{D} \rightsquigarrow \hat{C} \sqsubseteq E, E \sqsubseteq F, F \sqsubseteq \hat{D} \tag{NR2}$$
$$A \sqcap \hat{C} \sqsubseteq B \rightsquigarrow A \sqcap E \sqsubseteq B, \hat{C} \sqsubseteq E \tag{NR3}$$
$$A \sqsubseteq B \sqcap \hat{D} \rightsquigarrow A \sqsubseteq B \sqcap E, E \sqsubseteq \hat{D} \tag{NR4}$$
$$\exists R.\hat{C} \sqsubseteq B \rightsquigarrow \exists R.E \sqsubseteq B, \hat{C} \sqsubseteq E \tag{NR5}$$
$$A \sqsubseteq \exists R.\hat{D} \rightsquigarrow A \sqsubseteq \exists R.E, E \sqsubseteq \hat{D} \tag{NR6}$$

where $A, B \in BN_{\mathcal{T}}, \hat{C}, \hat{D} \notin BN_{\mathcal{T}}$, and E, F are fresh concept names introduced to define complex concept description. Note that $BN_{\mathcal{T}}$, as described in Section 2.4.1, is the smallest set of concepts containing: (i) the top concept ($\top$); (ii) all concept names A of $\mathcal{T}$ in N_C.

In addition, we can transform n-ary conjunctions of concept names, which is represented as $\sqcap_i A_i$ by applying the following equivalence transformation to the axiom:

Then $\mathcal{T}'$ consists of GCIs in the following forms:

$$A \sqsubseteq B \qquad \sqcap_i A_i \sqsubseteq B$$
$$A \sqsubseteq \exists R.B \qquad \exists R.A \sqsubseteq B$$

where A, A_i, B are concept names.

3.2.2 Completion Rules

This algorithm works on a data structure called *completion sets* (Baader, Brandt, & Lutz, 2005; Turhan, 2010). Secondly, the algorithm will map $\mathcal{T}'$ to completion sets. There are two types of completion sets used in this algorithm:

- $\mathcal{S}(A)$ for each concept name A appearing in $\mathcal{T}'$
- $\mathcal{R}(R)$ for each role name R appearing in $\mathcal{T}'$

In the algorithm, both types of completion sets are initialised as the following:

- $\mathcal{S}(A) := \{A, \top\}$ for each concept name A appearing in $\mathcal{T}'$
- $\mathcal{R}(R) := \emptyset$ for each role name R appearing in $\mathcal{T}'$

Thirdly, the algorithm applies the completion rules in Figure 3.4 to the completion sets until no more rule can be applied. Finally, the algorithm reads subsumption relationships.

Rules	Expression
CR1	If $A_1, ..., A_n \in \mathcal{S}(C), \sqcap(A_1, ..., A_n) \sqsubseteq B \in \mathcal{T}$, and $B \notin \mathcal{S}(C)$ then $\mathcal{S}(C) := \mathcal{S}(C) \cup \{B\}$
CR2	If $A \in \mathcal{S}(C), A \sqsubseteq \exists R.B \in \mathcal{T}$, and $(C, B) \notin \mathcal{R}(R)$ then $\mathcal{R}(R) := \mathcal{R}(R) \cup \{(C, B)\}$
CR3	If $(C, D) \in \mathcal{R}(R), A \in \mathcal{S}(D), \exists R.A \sqsubseteq B \in \mathcal{T}$, and $B \notin \mathcal{S}(C)$ then $\mathcal{S}(C) := \mathcal{S}(C) \cup \{B\}$

Figure 3.4: Completion rules for $\mathcal{EL}$ (Suntisrivaraporn, 2009).

The rule **CR1** is used to handle GCIs in the forms of $A \sqsubseteq B$ and $\sqcap_i A_i \sqsubseteq B$. The rule **CR1** ensures that if a concept A implies B w.r.t. $\mathcal{T}'$ and A is in the completion set of any concept, B has to be in those completion sets as well. Similarly, if a conjunction $\sqcap_i A_i$ implies B w.r.t. $\mathcal{T}'$ and the conjuncts A_i are in the completion set of any concept C, B has to be in those completion sets. The rule **CR2** is used to handle GCIs in the from of $A \sqsubseteq \exists R.B$. The rule **CR2** ensures that if a concept A implies an existential restriction w.r.t. $\mathcal{T}'$ and A is in the completion sets of any concept C, then C implies the existential restriction. The rule **CR3** is used to handle GCIs in the from of $\exists R.A \sqsubseteq B$. The rule **CR3** ensures that if an existential restriction $\exists R.A$ implies a concept B w.r.t. $\mathcal{T}'$, A is in the completion set of any concept D, and $C \sqsubseteq_{\mathcal{T}'} \exists R.D$, then B has to be in the completion set of C. This is because an existential restriction $\exists R.A$ implies a concept B w.r.t. $\mathcal{T}'$ and A is in the completion set of any concept D implies $\exists R.D \sqsubseteq_{\mathcal{T}'} \exists R.A$ and $\exists R.D \sqsubseteq_{\mathcal{T}'} B$. Then we have $C \sqsubseteq_{\mathcal{T}'} \exists R.D$ and $\exists R.D \sqsubseteq_{\mathcal{T}'} B$. Therefore, $C \sqsubseteq_{\mathcal{T}'} B$.

The intuition is that these completion rules make implicit subsumption relationships explicit in the following sense (Baader, Lutz, & Suntisrivaraporn, 2005; Turhan, 2010):

- $B \in \mathcal{S}(A)$ implies that $A \sqsubseteq_{\mathcal{T}'} B$
- $(A, B) \in \mathcal{R}(R)$ implies that $A \sqsubseteq_{\mathcal{T}'} \exists R.B$

In addition, it can be said that $\mathcal{S}(A)$ contains subsumers or super concepts of A and $\mathcal{R}(R)$ contains only concept names, which subsumes A with existential restriction of R. The algorithm is terminated after a polynomial number of steps. Then the subsumption relationship between concept names A and B can be checked whether $B \in \mathcal{S}(A)$. The algorithm can be done in polynomial time (Brandt, 2004).

As a result, it can be seen clearly that the polynomial-time subsumption algorithm is more effective for $\mathcal{EL}$ than the tableau-based algorithms because of the high worst-case complexity of the tableau-based algorithms. Moreover, the polynomial-time subsumption algorithm does not need to transform subsumption to disjunction, which is the source of high complexity.

At first, the algorithm has been implemented in the CEL ontology reasoner initially (Baader et al., 2006). CEL is reintroduced as an OWL 2 EL reasoner (Mendez & Suntisrivaraporn, 2009). Over the years, this type of algorithms has been further studied and improved. One of the results of this study is *consequence-based procedures* (Kazakov et al., 2014). This procedure is implemented in the ELK reasoner (Kazakov et al., 2014). This procedure applies the completion rules in a goal-directed way, which means that it avoids redundant inferences. This makes this procedure more efficient than the original procedure. This procedure also has several distinguished properties such as optimal worst-case complexity and determinism, which means that this does not make choices or backtracking. In addition, this procedure has been improved to support $\mathcal{ALC}$ (Simancik, Kazakov, & Horrocks, 2011) and Horn-$\mathcal{SHIQ}$ (Kazakov, 2009). The improvements and extensions of this procedure can be found in (Kazakov, Krötzsch, & Simancík, 2011; Kazakov, Krötzsch, & Simancik, 2011; Kazakov, Kroetzsch, & Simancik, 2012; Kazakov & Klinov, 2014a, 2014b).

Chapter 4

Constraint Programming

The long-standing research in *Constraint Programming* (CP) (Rossi, Beek, & Walsh, 2006) has made tremendous progress. Implementations of CP-based techniques are referred as *Constraint Solvers.* Constraint Programming can support very generic forms of problems because Constraint Programming supports a wide range of constraints defined over finite domains such as numerical constraints, linear constraints, and Boolean constraints. One of the advantages of Constraint Programming is model (problem) and solver are clearly separated. The problem that we want to solve is defined declaratively, consisting of variables and constraints, in a high-level way. Then this high-level model is solved by a constraint solver. Advanced techniques such as Constraint Propagation (Bessiere, 2006) in CP have increased the number of problems that can be handled dramatically. Constraint Propagation can highly prune the search tree by maintaining consistencies during the search. Recently, a new approach to CP solving called *Lazy Clause Generation* (LCG) (Ohrimenko et al., 2007) has been presented. The main idea of this approach is to combine the advantages of *Boolean Satisfiability* (SAT) (Biere, Heule, van Maaren, & Walsh, 2009) and CP together to make a powerful solver. Gathering from the idea of the LCG approach, we believe that the powerful search and reasoning techniques offered by CP can be exploited to tackle the sources of inefficiency of tableau-based algorithms and improve the performance of ontology reasoning significantly.

In this chapter, the basic background notions of CP techniques, which are the baseline of the novel approach for reasoning about Description Logic in this thesis, are described. However, we do not go into the detail of specific theories of CP techniques. The main techniques and technologies of Constraint Programming that we have exploited in this work are described instead. In addition, the background of modelling that we exploited is provided.

Firstly, we start by providing the basic notations of Constraint Programming. Then we introduce some basic solving techniques for Constraint Programming such as *Search* (van Beek, 2006) and *Constraint Propagation* (Bessiere, 2006; Fruhwirth & Abdennadher, 2006), which is a well-known approach used in Constraint Programming. Next, we move to the modelling language called MiniZinc (Nethercote et al., 2007), which we use to model Description Logic to be able to solve it using CP. Finally, we close this chapter by explaining some advanced modelling and solving techniques that we exploited.

4.1 Basics on Search and Constraint Propagation

Constraint programming is a powerful approach for solving combinatorial problems and it offers efficient techniques in a wide range of domains and applications such as scheduling, planning, vehicle routing, computer networks, and bioinformatics (Rossi, van Beek, & Walsh, 2006). CP supports many forms of constraints such as numerical constraints, Boolean constraints, and constraints defined over finite domains. The basic idea of constraint programming is to solve a problem by satisfying given constraints in that problem. A combinatorial problem consists of a set of decision variables and a set of constraints, which are relations among the decision variables. Such a combinational problem is called *Constraint Satisfaction Problem* (CSP) (Freuder & Mackworth, 2006). The goal of Constraint Programming is to find a solution to a CSP that satisfies all constraints by assigning a value to each variable.

Generally, a *Constraint Solver* is any procedure that is able to solve such a problem. Most constraint solvers implement a well-known constraint solving algorithm that is so-called *Constraint Propagation.* However, Constraint Propagation is not complete. Therefore, it has to be interleaved with a *search algorithm* to achieve completeness. The first search algorithm was introduced for Boolean Satisfiability (SAT) problems (Davis, Logemann, & Loveland, 1962). Subsequently, the search algorithm has been extended to handle generic forms of problems.

4.1.1 Basics on CP and Notation

In this section, the *Constraint Satisfaction Problem* (CSP) (Freuder & Mackworth, 2006), which is the main problem of CP, is defined. The constraint satisfaction problem consists of:

- a finite set of *variables*
- for each variable, a *domain*, which is a finite set of possible values
- and a set of constraints, each of which is a relation over a set of variables.

Then let us formally define the constraint satisfaction problem.

Definition 4.1.1. Let X be a finite set of variables, i.e., $\{x_1, x_2, ..., x_n\}$, V be a finite set of values, i.e., $\{v_1, v_2, ..., v_m\}$. D is a set of domains $\{D(x_1), D(x_2), ..., D(x_n)\}$ for X such that $D(x_i) \subseteq V$. $D(x_i)$ is the set of possible values of x_i. A domain $D(x_i)$ is *stronger* than a domain $D'(x_i)$ if $D(x_i) \subseteq D'(x_i)$ for all $x_i \in V$. It can be written as $D(x_i) \sqsubseteq D'(x_i)$. C is a set of constraints $\{c_1, c_2, ..., c_t\}$. A constraint c_i is a pair $(scope(c_i), rel(c_i))$, where $scope(c_i) = \{x_i, x_2, ..., x_s\}$ is the *constraint scope*, which is an ordered subset of X, and $rel(c_i)$ is the *constraint relation*, which is a subset of the Cartesian product of the domains of variables in $scope(c_i)$ $D(x_1) \times D(x_2) \times ... \times D(x_s)$ that specifies the allowable combinations of values of the variables in $scope(c_i)$.

A Constraint Satisfaction Problem (CSP) is a triple $\mathcal{P} = \langle X, D, C \rangle$. A *solution* of CSP $\mathcal{P}$ is a n-tuple $sol(\mathcal{P}) = \langle a_1, a_2, ..., a_n \rangle$ such that $a_i \in D(x_i)$ and each constraint c_i is satisfied in the relation $rel(c_i)$, which is the projection of $sol(\mathcal{P})$ onto $scope(c_i)$. An *assignment* is written as $x_i = v_i$, which means that a variable x_i is assigned to a value v_i. Let $l = \{x_{i_1} = v_{i_1}, x_{i_2} = v_{i_2}, ..., x_{i_p} = v_{i_p}\}$ be a set of assignments and $scope(l) = \{x_{i_1}, x_{i_2}, ..., x_{i_p}\}$ be the set of variables involved in l. A *complete assignment* is an l such that $scope(l) = X$. A constraint c_i is satisfied by a set of assignments l such that $scope(c_i) \subseteq scope(l)$ if l specifies an allowable combination of values that the variables in $scope(c_i)$ can be assigned. Therefore, a solution $sol(\mathcal{P})$ of CSP $\mathcal{P}$ is a complete assignment such that all constraints are satisfied.

Next, we will use Example 4.1.1 to illustrate a CSP instance.

Example 4.1.1. Given a CSP $\mathcal{P}$ as follows:

- the set of variables $X = \{x_1, x_2, x_3\}$
- domains $D(x_1) = \{1, 2\}, D(x_2) = \{0, 1, 2, 3\}, D(x_3) = \{2, 3\}$

- the set of constraints $C = \{x_1 \neq x_2 \neq x_3, x_1 > x_2, x_1 + x_2 = x_3\}$

This CSP $\mathcal{P}$ has 3 variables: x_1, x_2, and x_3. In addition, each variable x_i is associated with $D(x_i)$. There are three constraints. The first constraint $x_1 \neq x_2 \neq x_3$ states that all variables must have different values. The second constraint $x_1 > x_2$ states that the value of x_1 must be greater than the value of x_2. The third constraint $x_1 + x_2 = x_3$ states that adding the value of x_1 to the value of x_2 must be equal to the value of x_3. One possible solution of this CSP is the set of assignments $\{x_1 = 2, x_2 = 1, x_3 = 3\}$. $\square$

Generally, the complexity of solving the constraint satisfaction problem is NP-COMPLETE (Feder & Hell, 2006). This means it is unlikely that there exists an algorithm to find a solution in polynomial time.

4.1.2 Search

After a CSP is modelled, there are many techniques that can be used to solve it (Freuder & Mackworth, 2006). A *backtracking* (BT) search is the most common algorithm that is used to solve CSP systematically (van Beek, 2006). A backtracking search algorithm performs a depth-first traversal of a search tree to find a solution to a CSP. In a simple formal backtracking search, each node of the search tree is corresponding to variable assignment decision. The search tree is generated as the search progresses. When a node in the search tree is visited, branches are built. Each branch represents alternative choices that may need to be examined in order to find a solution. The method of generating a node of the search tree is usually called *branching strategy* (van Beek, 2006), and several alternatives have been proposed and examined in the literature. After a node is generated, constraints are used to check whether that node may lead to a solution of the CSP and to prune subtrees that do not lead to a solution. If a node in the search tree does not lead to a solution, that node is called a *dead-end*. Backtracking search algorithms are typically *complete* search methods for constraint satisfaction problems. They guarantee that a solution will be found if there exists one, or show that there is no solution for a CSP.

More precisely, basic backtracking search tries to build a partial solution by assigning values to variables until it reaches a dead-end. This means that that partial solution cannot be further extended consistently. Since there is branching during search process, when it reaches a dead-end, it cancels the last decision and tries another branch. This search is done systematically in order to guarantee that all possible branches will be tried. The search also checks whether a new choice (candidate solution) made satisfies the respective constraints one by one instead of checking after a complete assignment is generated for all variables.

Generally, in a simple formal backtracking search, the representation of the backtracking search process is a search tree, where each node after the root node represents an assignment of a value to a variable and each branch represents a candidate partial solution. If a partial solution is found to be unextended, a subtree corresponding to that partial solution can be pruned. When a node is visited during the backtracking search process, only constraints that have variables instantiated (constraints that have the current decision variable and the past variables along a branch) are checked at that node. If constraints are not satisfiable, the next choice of value of the current variable is tried. If there are no values of the current variable left that can satisfy all constraints, the backtracking search backtracks to the most recently instantiated variable. When the last variable is instantiated and all constraints are satisfied, a solution is found.

Let us use Example 4.1.1 to illustrate a search tree built by the backtracking search algorithm as shown in Figure 4.1. Recall that the problem in Example 4.1.1 requires a complete assignment of three variables x_1, x_2, x_3, where their domains are $D(x_1) = \{1, 2\}, D(x_2) = \{0, 1, 2, 3\}, D(x_3) = \{2, 3\}$ to satisfy the constraints $x_1 \neq x_2 \neq x_3, x_1 >$

$x_2, x_1 + x_2 = x_3$. As shown in Figure 4.1, the root node at level 0 is the empty set of assignments and a node at level i is a set of assignments $x_1 = v_1, .., x_i = v_i$. When the backtracking search reaches a dead-end (the red node in the search tree), that means the current assignments do not satisfy the constraints and the subtrees are pruned. Then the backtracking search backtracks and tries the next value of the current varaible. For example, at level 2, when assignments $x_1 = 1, x_2 = 1$ do not satisfy the constraint $x_1 > x_2$, a backtracking process occurs and the next value, which is 2 is tried. Note that this search tree performed under the assumption that the instantiation is in a static order, where variable x_i is always selected at level i in the search tree and values are assigned to the variables in the order of domain of each variable x_i.

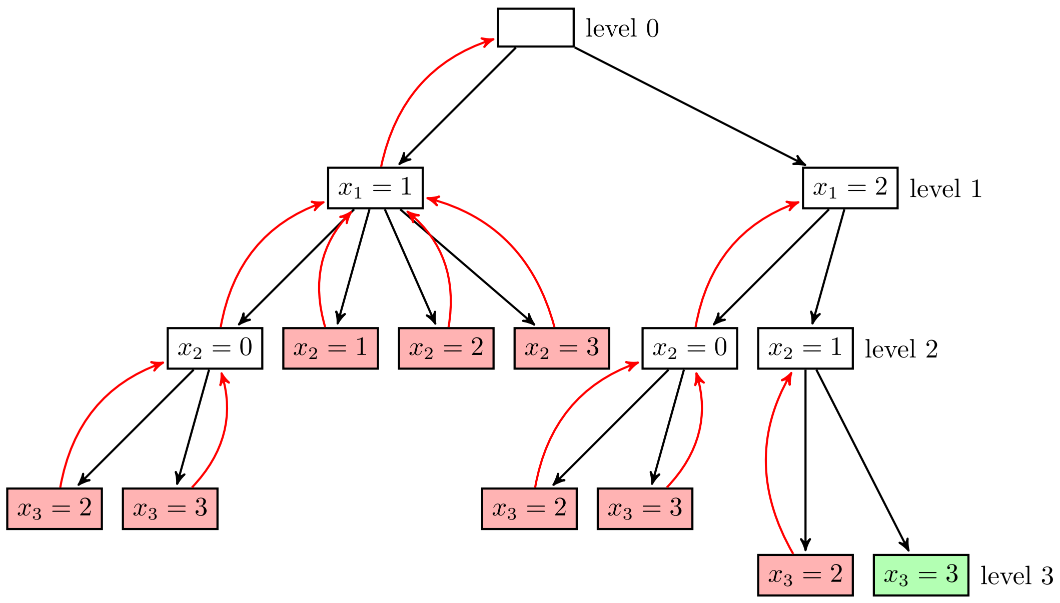


Figure 4.1: A search tree of backtracking search of Example 4.1.1.

A more general version of backtracking search does not only assign values to variables on each node, but also can split a problem into two subproblems by adding constraints on branches. For example, given a CSP $\mathcal{P} = \langle X, D, C \rangle$ from Example 4.1.1, a backtracking search can split $\mathcal{P}$ into two subproblems $\mathcal{P}_1 = \langle X, D, C \cup \{x = 1\} \rangle$ on one branch and $\mathcal{P}_1 = \langle X, D, C \cup \{x \neq 1\} \rangle$ on another branch. Then the search algorithm explores $\mathcal{P}_1$ until a solution is found. If there is no solution to $\mathcal{P}_1$, the search then backtracks to $\mathcal{P}_2$. If there is no solution to either $\mathcal{P}_1$ or $\mathcal{P}_2$, $\mathcal{P}$ has no solution.

The main drawback of backtracking search is *thrashing*. A dead-end is reached due to the same reason multiple times since backtracking search does not identify the real reason of the conflict. As can be seen from Figure 4.1, the branches, $\{x_1 = 1, x_2 = 1\}$, $\{x_1 = 1, x_2 = 2\}$, and $\{x_1 = 1, x_2 = 3\}$, reach a dead-end because of the constraint $x_1 > x_2$, which states that a value of x_1 must be greater than a value of x_2. In addition, backtracking search may not terminate within polynomial time. Due to these reasons, backtracking search has been improved in order to maximise its practical efficiency.

One of the improvements is *Backjumping* (BJ) (Gaschnig, 1979). The main difference between backtracking algorithm and backjumping algorithm is that if backjumping algorithm finds an inconsistency, it identifies the source of inconsistency by analysing the situation. It uses the unsatisfiable constraints to find the conflicting variable. Then it will backjump to the most recent conflicting variable. In addition, the improvement has been done by equipping backtracking search with constraint propagation (Bessiere, 2006) techniques to remove inconsistent values at the early state of the search, or making use of search heuristics to guide search effectively (van Beek, 2006). The techniques are not always independent and the search can be improved significantly by combining two or more techniques. Robust backtracking search algorithms, which can solve large and challenging practical problems, can be obtained by the best combinations of these techniques.

4.1.3 Constraint Propagation

Thrashing behaviour always happens when backtracking search algorithms are used to solve CSP problems (Bobrow & Raphael, 1974; Freuder & Mackworth, 2006). In order to minimise thrashing behaviour, *Constraint Propagation* (Bessiere, 2006; Fruhwirth & Abdennadher, 2006) is used. Constraint propagation is a technique used to maintain *Local Consistency* (Bessiere, 2006), which can identify and eliminate thrashing behaviour, during backtracking search. Constraint propagation makes implicit constraints explicit and propagates information contained in that constraint to the other constraints. This can dramatically reduce the search space that need to be searched by removing many dead-ends.

For Constraint Propagation (Bessiere, 2006; Fruhwirth & Abdennadher, 2006), the main idea is to reduce the size of a problem by reducing the size of domains of the variables until all variables have only one value (solution), which satisfies all constraints. As mentioned earlier, Constraint propagation is incomplete. Therefore, Constraint propagation has to be combined with search to provide a complete solution procedure. Constraint propagation propagates the information contained in one constraint to the neighbouring constraints. The information is domain changes of each variable of the constraint. It removes the inconsistent values from the domains of the variables of the constraints. This process is performed repeatedly until there is no domain change. This means that Constraint propagation reaches a *fixed-point*. The process of this technique aims to transform a CSP problem into an equivalent one that may be easier to solve without loosing solutions of the problem. The entities that perform constraint propagation on a constraint are called *propagators*. A propagator f is a monotonically decreasing function from domains to domains: $f(D(x_i)) \sqsubseteq D(x_i)$.

Local Consistency is a property that needs to be maintained on the constraints of a problem in order to detect and eliminate inconsistent partial assignments. Local consistencies are typically combined with backtracking search algorithms in order to remove some but not all incompatible values from domains of variables. The simplest consistency technique that can be used on a CSP is called *Node Consistency* (NC) (Bessiere, 2006). This consistency technique only supports unary constraints. A CSP is node consistent if and only if all values in a domain of a variable satisfy the unary constraints of that variable. If a domain of a variable contains a value that does not satisfy the unary constraints on that variable, the instantiation of that variable always lead to an immediate failure. In easy words, solutions do not have that value. Therefore, the node consistency can be obtained by removing those values from the domain of that variable that do not satisfy the unary constraints. For example, given a CSP that contains the constraint $c_1 : x < 10$, the domain of x is $D(x) = [1..20]$. x is not node consistent with respect to c_1. As a result, the propagation of c_1 will remove values [10..20] from $D(x)$.

The most common and popular local consistency is *Arc Consistency* (AC) (Mackworth, 1977a; Bessiere, 2006). In contrast to node consistency, arc consistency is used to guarantee consistency of binary constraints. The main idea of arc consistency aims to ensure that all values in a domain are consistent with respect to every constraint. In a constraint graph, binary constraints are considered as arcs (directed edges). A binary constraint is a relation on two variables. An arc (x_i, x_j) is *arc consistent* if and only if for every value v_i in the domain of x_i, which satisfies the constraints on x_i, there exists some values v_j in the domain of x_j such that $x_i = v_i$ and $x_j = v_j$ are permitted by the binary constraint between x_i and

x_j. Since an arc is considered as a directed edge in constraint graph, if an arc (x_i, x_j) is arc consistent, it does not mean that an arc (x_j, x_i) is consistent. A CSP is *arc consistent* if and only if every arc (x_i, x_j) in its constraint graph is arc consistent. Currently, modern constraint solvers use non-binary constraints, such as linear constraints or all different, and that the notion of arc consistency can be easily generalised to the non-binary case. This is called *Generalised Arc Consistency* (GAC) (Mackworth, 1977b; Bessiere, 2006). In addition, many propagators for constraints (e.g. for linear constraints) do not actually perform arc consistency but *Bound Consistency* (J. Puget, 1998).

Let us use Example 4.1.1 to illustrate a search tree built by the backtracking search algorithm combined with Constraint propagation as shown in Figure 4.2. Recall that the problem in Example 4.1.1 requires a complete assignment of three variables x_1, x_2, x_3, where their domains are $D(x_1) = \{1, 2\}, D(x_2) = \{0, 1, 2, 3\}, D(x_3) = \{2, 3\}$ to satisfy the constraints $x_1 \neq x_2 \neq x_3, x_1 > x_2, x_1 + x_2 = x_3$. At level 1 of the search tree in Figure 4.2, the variable x_1 is assigned to 1. The values 1, 2, and 3 are removed from $D(x_2)$ of variable x_2 according to the constraint $x_1 > x_2$. In addition, the values 2 and 3 are removed from from $D(x_3)$ of variable x_3 due to the constraint $x_1 + x_2 = x_3$. As can be seen, since the domain $D(x_3)$ is empty, the search tree reaches a dead-end and the next value of variable x_1 will be tried. Next, the variable x_1 is assigned to 2. The values 2 and 3 are removed from $D(x_2)$ of variable x_2 according to the constraint $x_1 > x_2$. The value 2 is removed from $D(x_3)$ of variable x_3 due to the constraint $x_1 \neq x_2 \neq x_3$. Then the value 0 is removed from $D(x_2)$ of variable x_2 according to the constraint $x_1 + x_2 = x_3$. Then there is only one value in the domains of each variables. As a result, this is a solution. Note that this search tree performed under the assumption that the instantiation is in a static order, where variable x_i is always selected at level i in the search tree and values are assigned to the variables in the order of domain of each variable x_i. Considering the search trees in Figure 4.1 and Figure 4.2, the number of fails is reduced from 8 (the search tree of backtracking search without constraint propagation) to 1 (the search tree of backtracking search with constraint propagation). This means that Constraint propagation can improve the efficiency of the backtracking search.

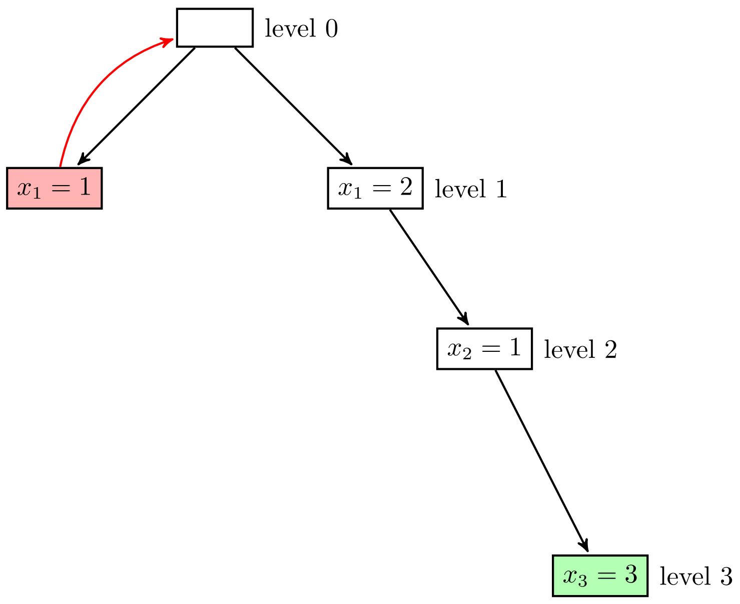
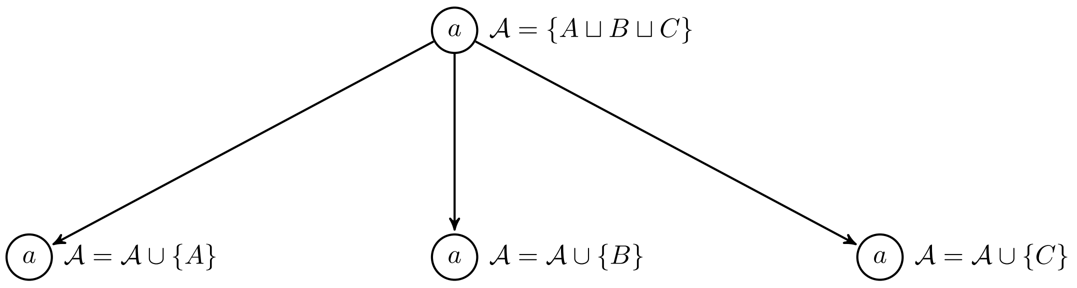


Figure 4.2: A search tree of backtracking search combined with constraint propagation of Example 4.1.1.

As can be seen, there are many techniques that we can exploit to improve the performance of DL reasoning especially dealing with OR-branching. Let us use Example 4.1.2 to illustrate the advantage of CP techniques comparing to tableau-based algorithm.

Example 4.1.2. Given a concept $C := A \sqcup B \sqcup D$, where A, B, and C are atomic concepts. Let $\mathcal{A}$ be a constraint system and a is an individual. Then we will use the tableau-based algorithm described in Section 3.1 and CP techniques to solve concept satisfiability problem.

Figure 4.3 shows that there are 3 branches, when the tableau-baed algorithm is applied to C. Then the tableau-baed algorithm needs to check every branch in order to check concept satisfiability of C. On the other hand, from Figure 4.4, CP starts by assigning A to be *true*. Then CP can find a solution, where C is satisfiable, using constraint propagation. One of solutions is $A = true, B = true, C = true$. CP do not perform branching and checking all branches in order to check concept satisfiability of C. In this case, CP use constraint propagation to deal with disjunction. As a result, CP can check concept satisfiability faster than the tableau-baed algorithm in this case.



a
$\mathcal{A} = \{A \sqcup B \sqcup C\}$
a
$\mathcal{A} = \mathcal{A} \cup \{A\}$
a
$\mathcal{A} = \mathcal{A} \cup \{B\}$
a
$\mathcal{A} = \mathcal{A} \cup \{C\}$

Figure 4.3: Solving concept satisfiability in Example 4.1.2 using tableau-based algorithm.

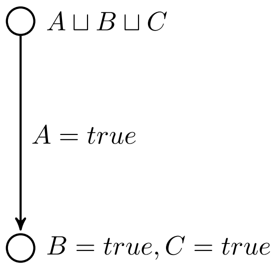


Figure 4.4: Solving concept satisfiability in Example 4.1.2 using CP techniques.

4.2 MiniZinc Modelling Language

As mentioned in Section 2.3, we normally use the word *model* for the interpretation that satisfies a concept or TBox in Description Logic. Therefore, the model for Constraint Programming that we will describe in this thesis is stated as *constraint model*. In order to solve a problem using Constraint Programming, the problem needs to be defined as a *constraint model*. There are many different modelling languages since each constraint solver is compatible with different modelling languages (Nethercote et al., 2007). This is very difficult to constraint modellers, who would like to conduct experiments using different solvers since they may need to learn new modelling languages.

Therefore, the modelling language that we exploited in this work is MiniZinc (Nethercote et al., 2007). MiniZinc is a medium-level declarative modelling language. In addition, MiniZinc has been developed as a standard modelling language for modelling Constraint Programming problems. It is also high-level and expressive enough to express most CP problems. As a standard modelling language, MiniZinc is solver-independent. This means that once the problem is modelled in MiniZinc constraint model of the problem can be solved by different constraint solvers. MiniZinc has been found useful in a number of applications such as Middleware for Adaptation of Web Service Compositions (Lu & Tosic, 2010), Portfolio Selection in finance and economics (de la Barra et al., 2013), and Data Mining (Guns et al., 2013).

Firstly, the basic specification that we exploited of MiniZinc such as types and expressions is provided. Then an example of how the specification of MiniZinc is close to the syntax and semantics of Description Logic is described.

4.2.1 Basic Specification of MiniZinc

Generally, a MiniZinc problem has two components. The first component is *constraint model*, which describes the structure of a class of problems. The second component is *data*, which specifies a particular problem within that class. In this thesis, only *constraint models* are considered. All specifications here are gathered from (Nethercote et al., 2007).

There are three scalar types: Booleans (specified as `bool`), integers (specified as `int`), and floats (specified as `float`), supported by MiniZinc. In addition, MiniZinc supports two compound types: sets (specified as `set`) and arrays (specified as `array`). In addition to types, MiniZinc supports two different *instantiations* (inst) of variables. A variable can be defined as either *parameter* (specified as `par`), which is a variable in the model that is fixed to a known value, or *decision variable* (specified as `var`), which is a variable with unknown value.

Those three scalar types, Booleans, integers, and floats, can be either parameter or decision variable. For instance, `var int:x;`, `bool:y;`, and `par float:z;` are variable of types, integers, Booleans, and floats respectively. Note that if the inst is omitted, the default is `par`. Any `par` variable can be initialised in MiniZinc constraint models. For example, `int:x = 1..5;` is an integer variable that needs one value in a range of 1 to 5.

Sets of integers can be either `par` or `var`. The other sets must be `par`. In addition, sets can contain only scalars which are `par`. For example, the declaration of the set variable `s1`, `var set of 1..5:s1;`, is allowed. However, the declarations of set variables `s2` and `s3`, `var set of {true, false}:s2;` and `var set of 1.5..5.5:s3;`, are not allowed since `y` and `z` are not integers.

Arrays can be only `par` variable with fixed length. As normal arrays in other programming language, they can have multi-dimension. Each dimension is defined by a contiguous range of integers. Arrays can contain `par` or `var` scalars or sets of integers. For example, `array[0..5,0..5] of var set of 1..5:a` is a 2-dimension array of integer decision variables.

In addition, *set ranges*, *set literals*, and `par` set variables can be used as types of variable. The types of variables is the same type of declared set and the value of variable is constrained to a member in the set. For example,

1. `0..5:x1;` means that the type of variable `x1` is integer and the value of `x1` is between 0 to 5,

2. `var {2,4,6}:x2;` means that the type pf variable `x2` is integer and the value of `x2` is in {2,4,6},

3. `var 0.2..1.5:x3;` means that the type pf variable `x3` is float and the value of `x3` is between 0.1 to 1.5.

There are several kinds of expressions that are supported by MiniZinc. Variable names are expressions and they can be written using alphabets and number, e.g. `x1` and `x2`. Scalar literals are written as usual such as `0`, `1.5`, and `true`. Sets are stated using set literals or set comprehensions. For example: `{0,1,2}` or `{i+j | i,j in 1..5 where i == j}`. Multiple variables per generator, multiple generators are allowed in set comprehensions. A filtering `where` clause is allowed in generators.

Similarly, Arrays are stated using array literals or array comprehensions. For example: `[0,1,2]` or `[i | i in 1..5]`. If-then-else expressions are allowed. For instance, we can use `if i > 0 then i = 1 else i = 0 endif` in a MiniZinc constraint model. Another important expression for this work is functions since they can be used to support aggregation (see Section 7). A function can be called as usual, e.g. `even(x)`. The function can be called with combining an array comprehensions. A *generator call* `P(Gs)(E)` is same as `P([E|Gs])`. Note that the parentheses around the `E` are compulsory to avoid possible ambiguity, where the generator call is part of large expression. For example, the function `sum` can be used as `sum(i,j in 1..5)(d[i,j])` or `sum([d[i,j] | i,j in 1..5])`.

MiniZinc supports many useful built-in operations. The operations that are important for this work include comparisons (e.g. `==`, `>=`), arithmetic operations (e.g. `sum`, `min`), logical operations (e.g. `\/`, `forall`), set operations (e.g. `intersect`, `union`, `subset`), and coercions (e.g. `bool2int`). These operations work with parameters and decision variables.

Now, we have presented all variable types, expressions and operations that are necessary for this work. Next, the relationship between MiniZinc and Description Logic will be presented.

4.2.2 MiniZinc and Description Logic

This section presents the relationship between Description Logics and CP models by encoding the Description Logic $\mathcal{EL}$ presented in Section 2.4.1 into MiniZinc. Regarding the specification in Section 4.2.1, we now show how the Description Logic $\mathcal{EL}$ can be translated to MiniZinc as follows:

- Individuals a, b are translated to integers, i.e., `1, 2`.

- The top concept $\top$ is translated to a set variable of integers `T`. `T` denotes the abstract domain or universe (a set of all individuals in the domain).

- A concept A is easily translated to a set variable of integers `A`.

- For concept conjunction $A \sqcap B$, the concept name A and B are translated to set variables of integers `A` and `B` respectively. Then the conjunction $\sqcap$ is translated to set operation `intersect`. As a result, the concept conjunction $A \sqcap B$ is translated to `A intersect B`.

- For existential restriction $\exists R.B$, the concept name B is translated to a set variable of integers `B`. The role R is translated to an array variable of set variables of integers `R`, where indexes of `R` are predecessor individuals and individuals in the set are R-successors. For example, for an array `R = [{2,3},{4,5}]`, this means that individuals `2, 3` are R-successors of the individual `1` and individuals `4, 5` are R-successors of the individual `2`. On the other hand, the individual `1` is predecessor of individuals `2, 3` and the individual `2` is predecessor of individuals `4, 5`. The semantics of existential restriction is that there exists at least one R-successor, which is an element in `B`. As a result, the existential restriction can be translated to either

  ```
  {i | i in T where exists(R[i] intersect B)}
  ```

  or

```
{i | i in T where card(R[i] intersect B)>=1}
```

where **exists** is an operation to check existence of elements in a set, **card** is an operation to get cardinality of a set, and `i` is an individual.

As can be seen, there are two way to translate the existential restriction. Based on our preliminary experience, the encoding based on the **exists** function is more efficient than using the **card** function. However, a more detailed evaluation of different encodings should be performed as future work.

- The concept inclusion $A \sqsubseteq B$ is translated to `A subset B` using the **subset** operation.
- The concept equivalence $A \equiv B$ is translated to `A = B`.

Let us use the following example to illustrate how to encode an $\mathcal{EL}$ TBox to a MiniZinc constraint model.

Example 4.2.1. Given an $\mathcal{EL}$ TBox $\mathcal{T}$ containing the following axioms:

$$\begin{aligned}\mathsf{GeneralisedStructure} &\sqsubseteq \mathsf{DomainCategory}\\ \mathsf{AbstractStructure} &\sqsubseteq \mathsf{GeneralisedStructure}\\ \mathsf{DiabetogenicStructure} &\equiv \mathsf{GeneralisedStructure} \sqcap\\ &\qquad \exists\ \mathsf{IsCausallyLinkedTo.Diabetes}\end{aligned}$$

Before we encode this TBox, we need to normalise $\mathcal{T}$ to the following TBox $\mathcal{T}'$ as described in Section 2.4.1:

$$\begin{aligned}\mathsf{GeneralisedStructure} &\sqsubseteq \mathsf{DomainCategory}\\ \mathsf{AbstractStructure} &\sqsubseteq \mathsf{GeneralisedStructure}\\ \mathsf{DiabetogenicStructure} &\equiv \mathsf{GeneralisedStructure} \sqcap A\\ A &\sqsubseteq \exists\ \mathsf{IsCausallyLinkedTo.Diabetes}\end{aligned}$$

A is a fresh introduced concept name. Then we can encode $\mathcal{T}'$ to the following MiniZinc constraint model:

```
GeneralisedStructure subset DomainCategory
AbstractStructure subset GeneralisedStructure
DiabetogenicStructure = GeneralisedStructure intersect A
A subset {i | i in T where
card(IsCausallyLinkedTo[i] intersect Diabetes) >= 1}
```

Note that we do not need to normalise the equivalence to subsumption since the equivalence can be translated directly to `=`. □

A summary of MiniZinc syntax of $\mathcal{EL}$ is presented in Table 4.1. In Table 4.1, due to the presentation, we will use some symbols that are very close to MiniZinc syntax instead of the actual MiniZinc syntax. For example, the **intersect** operation is represented as $\cap$.

The semantics of $\mathcal{EL}$ (and other DLs) is normally defined in terms of *interpretations* as described in Section 2.4.1. In order to explain the relationship between MiniZinc and Description Logic, we will describe the semantics using Set Theory.

The semantics of MiniZinc is defined in terms of *solutions.* MiniZinc is used to solve CSPs, for which solutions are well defined (Freuder & Mackworth, 2006). A solution $\mathcal{S}$ consists of a nonempty set $\mathtt{T}^{\mathcal{S}}$, namely the domain of $\mathcal{S}$, and a solution function $\cdot^{\mathcal{S}}$ that

Table 4.1: The MiniZinc syntax of $\mathcal{EL}$.

Concepts	**Syntax**	**MiniZinc Syntax**
top concept	$\top$	`T`
atomic concept	A	`A`
conjunction	$\hat{C} \sqcap \hat{D}$	$\hat{\texttt{C}} \cap \texttt{D}$
existential restriction	$\exists R.\hat{C}$	`{i	i in T where exists(R[i] ∩ Ĉ)}` or `{i	i in T where card(R[i] ∩ Ĉ)>=1}`
Axioms	**Syntax**	**MiniZinc Syntax**
concept inclusion (subsumption)	$\hat{C} \sqsubseteq \hat{D}$	$\hat{\texttt{C}} \subseteq \texttt{D}$
concept equivalence	$\hat{C} \equiv \hat{D}$	$\hat{\texttt{C}} = \texttt{D}$

maps each set variable of atomic concept (or concept name) `A` to a set $\texttt{A}^S \subseteq \texttt{T}^S$, each array variable of atomic role (or role name) `R` to a set of pairs of individuals $\texttt{R}^S \subseteq \texttt{T}^S \times \texttt{T}^S$, and each integer of individual name `a` to the corresponding element $\texttt{a}^S \in \texttt{T}^S$. The extensions of $\cdot^S$ to complex concepts and roles are defined in the *Semantics* column in Table 4.2, where $(\texttt{a}, \texttt{b}) \in \texttt{R}^S$ means that an individual `b` in $\texttt{T}^S$ is a *R^S-successor of a*. A concept description $\hat{C}$ is defined through the concept constructs listed at the top of Table 2.4. For example, $\hat{C} = A \sqcap B \sqcap C$ is encoded into $\hat{\texttt{C}} = \texttt{A} \cap \texttt{B} \cap \texttt{C}$. Then the semantics of $\hat{\texttt{C}}$ is defined as $\hat{\texttt{C}}^S = \texttt{A}^S \cap \texttt{B}^S \cap \texttt{C}^S$.

Table 4.2: The syntax and semantics of $\mathcal{EL}$ in terms of sets.

Concepts	**MiniZinc Syntax**	**Semantics**
top concept	`T`	$\texttt{T}^S$
atomic concept	`A`	$\texttt{A}^S$
conjunction	$\hat{\texttt{C}} \cap \texttt{D}$	$\hat{\texttt{C}}^S \cap \texttt{D}^S$
existential restriction	`{a	a in T where exists(R[a] ∩ Ĉ)}` or `{a	a in T where card(R[a] ∩ Ĉ)>=1}`	$\{\texttt{a} \in \texttt{T}^S \mid \exists \texttt{b} : (\texttt{a}, \texttt{b}) \in \texttt{R}^S \wedge \texttt{b} \in \hat{\texttt{C}}^S\}$
Axioms	**MiniZinc Syntax**	**Semantics**
concept inclusion (subsumption)	$\hat{\texttt{C}} \subseteq \texttt{D}$	$\hat{\texttt{C}}^S \subseteq \texttt{D}^S$
concept equivalence	$\hat{\texttt{C}} = \texttt{D}$	$\hat{\texttt{C}}^S = \texttt{D}^S$

The semantics in terms of sets is shown in Table 4.2. In Table 4.2, the **`intersect`** operation is represented as $\cap$ due to presentation reason.

In this section, the simple translation from Description Logic to MiniZinc is presented in order to give an idea about the relation between MiniZinc and Description Logic. With this MiniZinc translation, we can perform the reasoning services, consistency checking, concept satisfiability, and concept subsumption, as described in Section 2.3. More details of this translation will be given in Chapter 5 and 7.

4.3 Advanced Modelling and Solving Techniques

This section introduces some advanced modelling and solving techniques that we exploited. The first technique is called *Lazy Clause Generation* (LCG) (Ohrimenko et al., 2007). This technique is a CP advanced solving technique, which takes strength of SAT solving and CP solving techniques. The second technique is *Symmetry Breaking* (J.-F. Puget, 1993). This technique enables us to reduce *symmetry* (J.-F. Puget, 2005) in constraint models. The issue of symmetry is that it may require redundant search to solve constraint models. The final technique is *Search Heuristic* (van Beek, 2006). Since an ordering of decisions of variables in solving CSP, this technique helps us to improve the solving process.

4.3.1 Lazy Clause Generation

Recently, a new approach for CP solving has been proposed. This approach is called *Lazy Clause Generation* (LCG) (Ohrimenko et al., 2007). This approach combines the advantages of SAT solving such as efficient nogood learning and backjumping with the advantages of CP solving such as efficient constraint propagation and simple and powerful modelling. This leads to the development of an efficient learning CP solver. In the original model of Lazy Clause Generation (Ohrimenko et al., 2007), the constraint propagation engine (finite domain (FD) propagation) is built inside a SAT solver. That means a search is controlled by SAT and a constraint propagation engine is considered as a clause generator for SAT. The advantages of SAT solvers are:

- *Unit Propagation*: The propagation of SAT solvers is specialised for Boolean variables, with clausal constraints of the form $l_1 \vee l_2 \vee ... \vee l_n$ where l_i is a *literal*. A literal is a Boolean variable or its negation. Unit propagation can detect a conflict and simplify a clause regarding literals efficiently using watch literals.

- *Conflict Analysis* analyses the structure of unit propagation and generates new learnt clauses (nogoods) to ensure that the conflict will not happen again.

The original Lazy Clause Generation works as follows. The search is started and controlled by the SAT engine. After a decision is made, unit propagation is performed until it reaches a unit propagation fixpoint with a partial assignment. A domain $D(x_i)$ is generated from fixed literals in the assignment. Then an appropriate propagator is performed. If the domain generated by propagators $f(D(x_i))$ is different from the original domain $D(x_i)$, constraint propagators act as clause generators for a SAT solver. The constraint propagators generate clauses that explain the changes in domains instead of performing propagation to domains to obtain new domains. Each clause generated is added to the SAT solver. Then a new round of unit propagation is started. This continues until fixpoint when the next SAT decision is made. When a conflict is detected, an explanation of the conflict is constructed. If the explanation of failure is added to SAT engine, it will force SAT to fail and start nogood construction. After that, it backjumps to where the nogood would first propagate and the the domain $D(x_i)$ is reset back to its previous state.

The integer variables of the finite domain problem need to be represented as Boolean variables in order to achieve this. The integer variables x with $D(x) = [l..u]$, where $[l..u]$ is the range $\{d \in \mathrm{Z} | l \leq d \leq u\}$, can be represented as the Boolean variables $[[x = l]], ..., [[x = u]]$ and $[[x \leq l]], ..., [[x \leq u - 1]]$. The variable $[[x = d]]$ is true if the value d is taken by x. Otherwise, it is false. Similarly, the variable $[[x \leq d]]$ is true if a value less than or equal to d is taken by x. Otherwise, it is false. For integer variables with $D(x) = [0..1]$, they can be considered as a Boolean variable. In order to maintain consistency of domains, for any integer variable x, the clauses $DOM(x)$ that encode $[[x \leq d]] \rightarrow [[x \leq d+1]]$, where $l \leq d \leq u-1$ and $[[x = d]] \leftrightarrow [[x \leq d]] \wedge \neg[[x \leq d-1]]$ are added to the SAT solver.

Next, let us use Examples 4.3.1 and 4.3.2 to illustrate how the original lazy clause generation works.

Example 4.3.1. Consider a constraint $c = x_0 \leftrightarrow x_1 + 1 \leq x_2$ and the current domains are $D(x_0) = [0..1], D(x_1) = [1..10], D(x_2) = [-2..6]$. Assuming that unit propagation determine that $\neg[[x \leq 7]]$, this changes $D(x_1) = [1..10]$ to $D'(x_1) = [8..10]$. Then the propagator f for c is performed to the domain D'. It obtains $f(D'(x_0)) = 0$. The clausal explanation of the change in domain of x_1 is $\neg[[x_1 \leq 7]] \wedge [[x_2 \leq 6]] \rightarrow \neg x_0$. This explanation is added to the SAT solver as $[[x_1 \leq 7]] \vee \neg[[x_2 \leq 6]] \vee \neg x_0$. Domain $D''(x_0) = \{0\}$ is created due to this. Then the propagator f needs to be re-examined. □

Example 4.3.2. Consider constraints $C = \{x_0 \leq x_1, x_0 = x_2 + x_3, x_2 \geq 4 \vee x_3 \geq 5\}$ and the current domains are $D(x_0) = [1..10], D(x_1) = [1..10], D(x_2) = [2..5], D(x_3) = [3..5]$. Assume that the search adds the new clause $x_1 \leq 5$, which can be represented by $[[x_1 \leq 5]]$. The upper bound of x_0 is changed to 5 ($[[x_0 \leq 5]]$) with explanation $[[x_1 \leq 5]] \rightarrow [[x_0 \leq 5]]$ due to the constraint $x_0 \leq x_1$. The upper bound of x_2 is changed to 2 ($[[x_2 \leq 2]]$), and x_3 to 3 ($[[x_3 \leq 3]]$) with explanations $[[x_0 \leq 5]] \wedge \neg[[x_3 \leq 2]] \rightarrow [[x_2 \leq 2]]$ and $[[x_0 \leq 5]] \wedge \neg[[x_2 \leq 1]] \rightarrow [[x_3 \leq 3]]$ due to the constraint $x_0 = x_2 + x_3$. The constraint $x_2 \geq 4 \vee x_3 \geq 5$, which can be encoded to a clause $\neg[[x_2 \leq 3]] \vee \neg[[x_3 \leq 4]]$, makes $\neg[[x_3 \leq 4]]$ true. However, by the domain constraint, it makes $[[x_3 \leq 3]] \rightarrow [[x_3 \leq 4]]$ false. The explanation of this conflict (nogood) is $\neg[[x_3 \leq 2]] \wedge \neg[[x_2 \leq 1]] \wedge [[x_0 \leq 5]] \rightarrow false$. This explanation is constructed using the first unique implication point (1UIP) (Zhang, Madigan, Moskewicz, & Malik, 2001). □

However, the drawback of the original model is the search is not programmable and flexible. Therefore, the lazy clause generation has been re-engineered (Feydy & Stuckey, 2009). In the new model, a CP solver is a master solver and a SAT solver is considered as a global propagator constructed inside the CP solver. The advantages of the new model are the search is programmable, search strategies can be specified for a particular problem and it is more flexible than the original model. The evaluation of the new model also shows significant improvement. This technique has been implemented in CP solvers, Opturion CPX[1] and Chuffed[2].

4.3.2 Symmetry Breaking

A *Symmetry* of a CSP is a mapping of the CSP onto itself that preserves its structure or its solutions (J.-F. Puget, 2005). Symmetries occur naturally in many problems. In addition, symmetries may be introduced when a problem is modelled. Symmetry is considered a fundamental issue in constraint satisfaction since the symmetry may require a large amount of redundant search to solve a problem.

Solution symmetry (Cohen, Jeavons, Jefferson, Petrie, & Smith, 2006) is a permutation of the set of assignments that preserves the set of solutions. In other words, a symmetry is a bijection on the set of assignments that maps solutions to solutions and non-solutions to non-solutions. Therefore, for any solution symmetry σ, all assignments $A \in sol(\mathcal{P})$ if and only if $\sigma(A) \in sol(\mathcal{P})$.

We then introduce some common classes of symmetries, which are considered as special cases of solution symmetry (Gent, Petrie, & Puget, 2006). The first class is *variable symmetry*, which is a permutation on the variables that preserves solutions (J.-F. Puget, 2005). Therefore, if $\{x_i = v_i | i \leq i \leq n\}$ is a solution, then $\{x_{\sigma(i)} = v_i | i \leq i \leq n\}$ is a

[1]`http://www.opturion.com/cpx`

[2]`https://github.com/geoffchu/chuffed`

solution. The second class is *value symmetry*, which is is a permutation on the values that preserves solutions (J.-F. Puget, 2005). Therefore, if $\{x_i = v_i | i \leq i \leq n\}$ is a solution, then $\{x_i = \sigma(v_i) | i \leq i \leq n\}$ is a solution. Let us use Examples 4.3.3 and 4.3.4 to illustrate the variable symmetry and value symmetry respectively.

Example 4.3.3. Given a list of exams and time slots, we would like to assign time slots to exams. Each exam is taken by the same group of students. Therefore, two exams cannot be taken at the same time slot. The exams are considered as variables, i.e., exam1 is v_1, and the time slots are considered as values. Note that we only consider three exams in this example. Then we can formulate a CSP $\mathcal{P} = (X, D, C)$ as follows:

- the set of variable $X = \{x_1, x_2, x_3\}$
- the domains $D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3\}$
- the set of constraints $C = \{x_1 \neq x_2 \neq x_3\}$

One solution of this problem is $S = \{x_1 = 2, x_2 = 1, x_3 = 3\}$. Let us define the permutation p such that $p(x_1 = i) = (x_3 = i)$, $p(x_2 = i) = (x_1 = i)$, and $p(x_3 = i) = (x_2 = i)$ that interchanges the assignments involving varaibles x_1, x_2, and x_3. One of the permutation is $p(S) = \{x_1 = 1, x_2 = 3, x_3 = 2\}$. This permutation is a variable solution symmetry of $\mathcal{P}$ because this permutation maps solution to solution and it acts on the variables. $\square$

Example 4.3.4. Given a graph with 4 nodes as can be seen in Figure 4.5. We would like to colour the graph with three colours red, green, and blue. A condition is that no two adjacent nodes have the same colour. The nodes in the graph are considered as variables. The colours are considered as values. Then we can formulate a CSP $\mathcal{P} = (X, D, C)$ as follows:

- the set of variable $X = \{x_1, x_2, x_3, x_4\}$
- the domains $D(x_1) = D(x_2) = D(x_3) = \{red, green, blue\}$
- the set of constraint $C = \{x_1 \neq x_2, x_1 \neq x_3, x_2 \neq x_3, x_3 \neq x_4\}$

One solution of this problem is $S = \{x_1 = red, x_2 = green, x_3 = blue, x_4 = green\}$. Let us define the permutation p such that $p(x_i = red) = (x_i = blue)$, $p(x_i = green) = (x_i = red)$, and $p(x_i = blue) = (x_i = green)$ that interchanges the assignments involving values *red*, *green*, and *blue*. The permutation is $p(S) = \{x_1 = blue, x_2 = red, x_3 = green, x_4 = red\}$. This permutation is a value solution symmetry of $\mathcal{P}$ because this permutation maps solution to solution and it acts on the values. $\square$

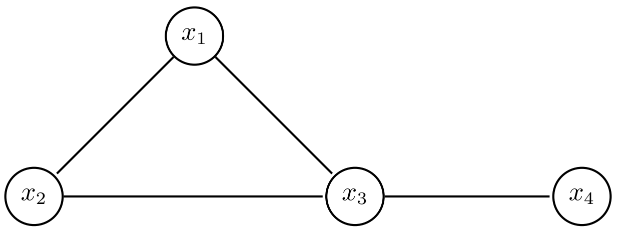
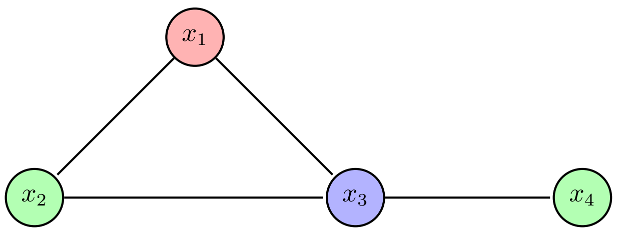
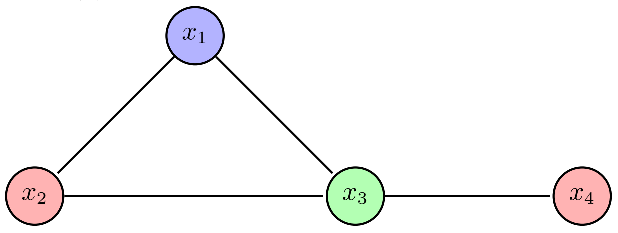


Figure 4.5: A graph with 4 nodes.

As described, symmetry is a fundamental issue of solving constraint satisfaction problems. It can increase the size of the search space. On the other hand, the symmetry can be used to reduce the size of the search for solutions of problems if the symmetry is eliminated. This is because a symmetry causes some subtrees of the search space to



(a) A solution of graph with 4 nodes.



(b) The value symmetry.

Figure 4.6: Solutions of colouring of the graph with 4 nodes.

be equivalent. For example, if one subtree has a solution, any symmetric subtree also has a solution. Similarly, if there is no solution in one subtree, there is no solution in any symmetric subtree. Thus, only one subtree needs to be explored if two subtrees are symmetrically equivalent.

Symmetry Breaking is an approach to reduce search effort by avoiding the exploration of redundant subtrees in the search space that are caused by symmetries. The purpose of symmetry breaking is to find a set of solutions where there is no other set of solutions, which is symmetrically equivalent. It has been proven that symmetries can be eliminated by adding constraints, which are so-called *symmetry breaking constraints*, to the original problem (J.-F. Puget, 1993). The modified problem can be solved more efficiently than the original problem. The symmetry breaking constraints are added to the problem before starting the search. These constraints prevent the exploration of some, or all of the redundant regions of the search space.

The most common technique to break variable symmetries was introduced by Crawford, Ginsberg, Luks, and Roy (1996). This technique is called *lex-leader*. Although the lex-leader technique is proposed for breaking variable symmetries, it can be used to break value symmetries as well (Petrie & Smith, 2003). The main idea of the lex-leader technique is to predefine one solution among the equivalence class of solutions under symmetry to be the canonical solution by adding constraints before starting search (Gent et al., 2006). These constraints are satisfied by only the canonical solutions. Regarding the lex-leader technique, variable symmetries can be eliminated by posting the constraint for each variable symmetry σ in order to ensure they are in lexicographic ordering. The constraint is in the following form:

$$[x_1, ..., x_n] \leq_{\text{lex}} [x_{\sigma(1)}, ..., x_{\sigma(n)}]$$

where x_1 to x_n is any fixed ordering on the variables. The lexicographic ordering is defined exactly as is standard in computer science. For example, $[x_1, x_2] \leq_{\text{lex}} [x_{\sigma(1)}, x_{\sigma(2)}]$ if and only if either $x_1 < x_{\sigma(1)}$ or $x_1 = x_{\sigma(1)}$ and $x_2 \leq x_{\sigma(2)}$.

Let us use Example 4.3.3 to illustrate the lex-leader technique. Recall that the problem in Example 4.3.3 requires a complete exam scheduling. Each exam is taken by the same group of students. Therefore, two exams cannot be taken at the same time slot. We can formulate this problem as shown in Example 4.3.3.

For this problem, there are 6 permutations (symmetries) on variables including the identity. We need to post 6 lex-leader constraints to break symmetries regarding the lex-leader technique as follows:

$$[x_1, x_2, x_3] \leq_{\text{lex}} [x_1, x_2, x_3] \tag{C1}$$
$$[x_1, x_2, x_3] \leq_{\text{lex}} [x_1, x_3, x_2] \tag{C2}$$
$$[x_1, x_2, x_3] \leq_{\text{lex}} [x_2, x_1, x_3] \tag{C3}$$
$$[x_1, x_2, x_3] \leq_{\text{lex}} [x_2, x_3, x_1] \tag{C4}$$
$$[x_1, x_2, x_3] \leq_{\text{lex}} [x_3, x_1, x_2] \tag{C5}$$
$$[x_1, x_2, x_3] \leq_{\text{lex}} [x_3, x_2, x_1] \tag{C6}$$

The set of lex-leader constraints can be simplified in order to remove redundant constraints without losing their power (Crawford et al., 1996; Luks & Roy, 2004). Let us demonstrate the simplification of the above set of lex-leader constraints. The constraint C1 can be immediately removed since it is always true. Considering the constraint C2, the first variable x_1 can be removed since it is clear that $x_1 = x_1$. Then we are left with $[x_2, x_3] \leq_{\text{lex}} [x_3, x_2]$. It can be written to $x_2 < x_3 \vee (x_2 = x_3 \wedge (x_3 \leq x_2))$. Next, $x_3 \leq x_2$ can be removed since $x_3 \leq x_2$ is always true if $x_2 = x_3$. As a result, the constraint C2 can be simplified to the constraint SC2 : $[x_2] \leq_{\text{lex}} [x_3]$ $(x_2 \leq x_3)$. Similarly, the constraint C3 can be simplified in the same way as the constraint C2 to the constraint SC3 : $[x_1] \leq_{\text{lex}} [x_2]$ $(x_1 \leq x_2)$.

Then let us consider the constraint C4. The constraint C4 can be written to $x_1 < x_2 \vee (x_1 = x_2 \wedge (x_2 < x_3 \vee (x_2 = x_3 \wedge x_3 \leq x_1)))$. This constraint can be immediately simplified to $x_1 < x_2 \vee (x_1 = x_2 \wedge (x_2 < x_3 \vee x_2 = x_3))$ since $x_3 \leq x_1$ is always true if $x_1 = x_2 = x_3$ and then it can be further simplified to $x_1 < x_2 \vee x_1 = x_2$ due to the constraint SC2 : $[x_2] \leq_{\text{lex}} [x_3]$. Since $x_1 < x_2 \vee x_1 = x_2$ is same as the constraint SC3, the constraint C4 can be removed.

Considering the constraint C5, $x_3 \leq x_2$ can be removed because $x_3 \leq x_2$ is always true if $x_2 = x_3$. Then it can be further simplified by removing $x_2 \leq x_3$ because of the constraint SC2. Since $\leq_{\text{lex}}$ is transitive, $[x_1] \leq_{\text{lex}} [x_3]$ can be implied by the constraints SC2 and SC3. As a result, the constraint C5 can be removed. Similarly, the constraint C6 can be removed according to the procedure described above. The final set of lex-leader constraints is as follows:

$$[x_2] \leq_{\text{lex}} [x_3] \tag{SC2}$$
$$[x_1] \leq_{\text{lex}} [x_2] \tag{SC3}$$

With these lex-leader constraints, the only solution that satisfies these constraints is $\{x_1 = 1, x_2 = 2, x_3 = 3\}$. The other permutations are excluded.

Some research has been done on the special case of row and column permutations in matrix models (Frisch, Jefferson, & Miguel, 2003). The lexicographical ordering has been shown that it can be used to break a subset of row or column symmetries of a matrix model (Flener et al., 2002). Moreover, some research has focused on symmetry breaking constraints using lexicographical ordering for handling symmetries on sets and other variables such as graph variables (Walsh, 2006).

4.3.3 Search Heuristics

A sequence of decisions (assignments) of which variable to be instantiated or branched and of which value to be assigned to the variable needs to be made, when a CSP is solved by backtracking search. Generally, these decisions are called *variable and value ordering* (van

Beek, 2006). The choice of variable and value ordering has been shown to be important for effectively solving many CSP problems (Ginsberg, Frank, Halpin, & Torrance, 1990; Bacchus & Van Run, 1995; Gent, MacIntyre, Presser, Smith, & Walsh, 1996). For example, Figure 4.1 shows the search of the problem in Example 4.1.1. It shows that there is no solution when the variable x_1 is assigned to 1. If the search was to try $x_2 = 1$ before $x_1 = 1$, then the solution would be found faster.

A variable or value ordering can be either *static*, where the order of variable or values is determined before the search starts and it is not changed thereafter, or *dynamic*, where the order of variable or values is determined at any point in the search process. If the ordering selected finds a solution or shows that there is no solution, where the fewest number of nodes are visited during the search, over all possible orderings, this variable or values ordering is said to be *optimal*.

Many variable orderings have been proposed. They can be classified into two groups: (1) heuristics that are based on the domain sizes of the variables and (2) heuristics that are based on the structure of the CSP.

Static variable ordering heuristics use only structural information of a problem as initially stated to determine the ordering. However, static variable ordering heuristics are considered weak since they miss valuable information and significant changes that occur during the search. Therefore, they are rarely used in practice. The known static variable ordering heuristics are the *min width* heuristic, which selects an ordering that has the minimum width over all ordering of the constraint graph (Freuder, 1982), and the *min bandwidth* heuristic, which selects an ordering that has the minimum bandwidth over all orderings of the constraint graph (Zabih, 1990). The bandwidth is the maximum distance (of edges) between two vertices. Another important static variable ordering heuristic is a simple input order. This heuristic allows us to specify an ordering of variables that need to be assigned with values.

The most common *dynamic* variable ordering heuristics is based on the *fail-first* principle (Haralick & Elliott, 1980), which is explained as "To succeed, try first where you are most likely to fail". Dynamic variable ordering heuristics consider the size of the domains of variables in order to determine which variable is the next one to be instantiated. The order of variable will be changed from one branch to another regarding the size of the domains since the backtracking search interleaved with constraint propagation prunes the size of the domains of variables from one branch to another according to the constraints. The first heuristic introduced by Golomb and Baumert in (Golomb & Baumert, 1965) and popularised by Haralick and Elliott in (Haralick & Elliott, 1980) is called `dom` or *minimum domain* heuristics. The main idea of `dom` is to select the next variable that has the smallest number of values remaining in its domain. This heuristic was shown that it work effectively with the forward checking algorithm (Haralick & Elliott, 1980). However, the main drawback of the simple `dom` heuristic is that the order of variables is rarely changed since variables often have the same size of domains in many problems at the beginning of the search.

In order to overcome the drawback mentioned above, the heuristic `dom/deg` is proposed (Bessiere & Régin, 1996). This heuristic selects a variable regarding the ratio of domain size of a variable over the degree of the variable, where the degree of the variable is the number of constraints, which involve the variable and at least one other unassigned variable. This heuristic selects the variable that has the minimal ratio.

The variable ordering heuristics `dom/wdeg`, which is one of the most efficient modern heuristics in Constraint Programming, is introduced and is shown to work well on many various problems (Boussemart, Hemery, Lecoutre, & Sais, 2004). In this heuristic, each constraint is associated with a weight, initially set to one. The associated weight is incremented when that constraint reaches a dead-end. Each variable is associated with a *weighted degree* (`wdeg`), which is the sum of the weights of all constraints that involve the variable and at least one other unassigned variable. Then the heuristic `dom/wdeg` selects the variable with the minimum ratio of the size of the domain of the variable over the weighted degree of the variable. Interestingly, the constraint propagation together with the heuristic `dom/deg` or `dom/wdeg` can reduce the need of backjumping on some problems (Boussemart et al., 2004; Lecoutre, Boussemart, & Hemery, 2004). Another important dynamic variable ordering heuristic is *Variable State Independent Decaying Sum* (VSIDS) (Moskewicz, Madigan, Zhao, Zhang, & Malik, 2001). This heuristic is used in SAT solvers and also in LCG solvers. In this heuristic, each variable is associated with a counter, which is initialised as 0. The counters of variable are incremented when a new clause that contains that variable, is added. At each decision point, a variable with the highest counter is selected to be assigned. Periodically, all counters are divided by a constant.

Once the decision to instantiate a variable is made, the values need to be examined. There are many value ordering heuristics. The static value ordering heuristic, which determines an order regarding an approximation of the number of solutions found by generating a tree relaxation of a problem is introduced (Dechter & Pearl, 1987). The heuristic, which selects the value that maximises the summation of the remaining domain sizes is proposed by Frost, Dechter, et al. (1995). However, Geelen (1992) claims that choosing the value that maximises the product of the remaining domain sizes is more effective since the value is likely to participate in a solution. Generally, the *succeed-fist* principle (Smith, 1996) suggests an idea that a value selected should be likely to be part of a solution.

Most modern constraint programming solvers such as Gecode[3], Opturion CPX[4], and Chuffed[5] provide several common variable ordering heuristics such as input order and first fail, and several value ordering heuristics such as ascending order, descending order or middle-out order. The variable ordering heuristics take a list of variables to be assigned during the search as parameter. The variables are assigned with values in the order that they are specified using *input order* heuristic. For instance, if the variables are $[x_1, x_2, ..., x_n]$, then x_1 is assigned first, then x_2, and so on. The *first fail* heuristic assigns values to variables that have the smallest size of domains. The size of domains is calculated during the search and constraint propagation. Some solvers also offer an option that user can customise an ordering of variables and values to suit the specific problem.

[3]`http://www.gecode.org/`

[4]`http://www.opturion.com/cpx`

[5]`https://github.com/geoffchu/chuffed`

Chapter 5

Encoding Reasoning Problems into MiniZinc

As described in Chapter 4, CP offers efficient search techniques and it supports a wide range of forms of constraints. These techniques have potentials to handle the main sources of inefficiency of the tableau-based algorithm such as disjunctions (OR-search). One approach to utilise the powerful search techniques of CP is to translate Description Logics to forms of constraints (formulas) that are supported by CP. Therefore, we propose an encoding scheme to encode Description Logics to MiniZinc models, which can be solved by CP. Our evaluations provides an evidence that our approach is competitive or even better than the dedicated tableau-based reasoners for some cases in benchmarks.

In the first step of our work, we start investigating the idea to encode the reasoning problems of $\mathcal{ALC}$ to MiniZinc models. Then we can exploit state-of-the-art CP solvers. We propose an efficient encoding approach and test it on an extensive set of benchmarks, comparing against the main state-of-the-art DL reasoners available. Our encoding scheme translates concepts, roles and features in Table 2.4 and 2.5, in a direct and succinct way, to set and array variables in MiniZinc respectively.

The rest of this chapter is structured as follows. In Section 5.1, we define our novel encoding scheme to encode $\mathcal{ALC}$ into CP problems. Then we prove that our encoding scheme is sound and complete in Section 5.2. Finally, we close this chapter by describing our evaluation setup and discussing the results in Section 5.3.

5.1 The Encoding of $\mathcal{ALC}$ and its sub-logics

In this chapter, we start exploring the idea of solving reasoning problem in Description Logics by translating them to MiniZinc models. Then such models can be handled by the modern CP solvers. We begin our investigation from the consistency checking problem in $\mathcal{ALC}$. In addition, concept satisfiability and concept subsumption can be reduced to the consistency checking problem (Baader & Nutt, 2003).

5.1.1 $\mathcal{ALC}$ to MiniZinc

This section presents the MiniZinc syntax and description. We will explain how MiniZinc syntax is close to the semantic of $\mathcal{ALC}$ regarding Table 2.5.

In our encoding scheme, *individuals* are encoded as positive integers. Moreover, we encode the top concept $\top$ (the superclass of all concepts) and a concept A as set variables of individuals `T` and `A` respectively. The set variable `T` contains all individuals in encoding. Hence, `A` is always a subset of `T` for all concepts A. The bottom concept $\perp$ can be encoded to empty set (`{}`). A role R is a binary relation that maps individuals of one concept (set) to individuals (successors) of another concept (set). It is encoded as an array of sets `R`, where indices `i` are individuals in one set and the set `R[i]` contains R-successors of individuals `i`.

In Section 4.2.2, the MiniZinc encoding for the Description Logic $\mathcal{EL}$ is presented. $\mathcal{EL}$ allows concept name, conjunctions, and existential restrictions. The Description Logic $\mathcal{ALC}$ is an extension of $\mathcal{EL}$. $\mathcal{ALC}$ allows three more constructs: (1) negation, (2) disjunction, and (3) universal restriction. In more details, we borrow the set notion to explain how MiniZinc models are closely related to the semantics of $\mathcal{ALC}$. Let us explain MiniZinc syntax regarding the $\mathcal{ALC}$ language constructs. We have described the MiniZinc syntax for the top concept ($\top$), the bottom concept ($\bot$), an atomic concept (A), conjunctions ($\hat{C} \sqcap \hat{D}$), and existential restrictions ($\exists R.\hat{C}$) in Section 4.2.2. Therefore, we will go through the remaining language constructs as follows:

- Concept negation ($\neg\hat{C}$): regarding the semantics of concept negation, the top concept ($\top$) is a set of all individuals in the domain ($\Delta^{\mathcal{I}}$). Concept negation is thus a set difference between $\top$ and $\hat{C}$. Then the concept negation can be encoded to `T - Ĉ`.
- Disjunction ($\hat{C} \sqcup \hat{D}$): the disjunction can be interpreted as a set union between $\hat{C}$ and $\hat{D}$. There is a set operation **`union`** in MiniZinc, which is equivalent to $\sqcup$ in $\mathcal{ALC}$. Hence, the disjunction can be encoded to `Ĉ` **`union`** `D`.
- Universal restriction ($\forall R.\hat{C}$): the universal restriction means that if there exist R-successors, those R-successors need to be member of a set $\hat{C}$. In other words, a set `R[i]` is a subset of $\hat{C}$. In MiniZinc, there is a set operation **`subset`**, which is equivalent to $\sqsubseteq$ in $\mathcal{ALC}$. Regarding the encoding of role R, the universal restriction can be encoded to `{i | i` **`in`** `T` **`where`** `R[i]` **`subset`** `Ĉ)}` for an individual `i`.

All concept construct encodings have been described above. The only components left are axioms. The axioms can be encoded as follows:

- Concept inclusion (subsumption) ($\hat{C} \sqsubseteq \hat{D}$): the concept inclusion can be interpreted as a set $\hat{C}$ being a subset of a set $\hat{D}$. As mentioned, there is a set operation **`subset`** in MiniZinc. Therefore, the concept inclusion can be encoded to `Ĉ` **`subset`** `D`.
- Concept equivalence ($\hat{C} \equiv \hat{D}$): the concept equivalence can be interpreted as a set $\hat{C}$ being equivalent to a set $\hat{D}$. Thus, the concept equivalence can be encoded to `Ĉ = D`.

Table 5.1 presents the summary of $\mathcal{ALC}$ syntax together with MiniZinc syntax. Due to the presentation, we will use some symbols that are very close to MiniZinc syntax instead of the actual MiniZinc syntax. For example, the symbol $\subseteq$ is used to represent the **`subset`** operation.

Our encoding scheme closely follows the semantics of Description Logics. For example, the semantics of the existential restriction ($\exists R.B$) in Table 2.5 states that there exists some R-successor of an individual `i` and the R-successor is a member of B. The existential restriction is encoded into `{i | i` **`in`** `T` **`where card`**`(R[i]` **`intersect`** `B)>=1}` , which means that there exists at least one individual in the intersection of B and the set of R-successors of individual `i`. Note that MiniZinc offers the function **`exists`**, which has the same meaning as existential restriction. Therefore, existential restriction ($\exists R.B$) can be encoded to `{i | i` **`in`** `T` **`where exists`**`(R[i]` **`intersect`** `B)}` as well. We will explain this in detail later. Our encoding approach only supports *acyclic* TBoxes, where there are no cyclic dependencies between its concept names, i.e., concept names are neither defined

Table 5.1: Summary of syntax of $\mathcal{ALC}$ and MiniZinc syntax.

Concepts	**Syntax**	**MiniZinc Syntax**
top concept	$\top$	`T`
bottom concept	$\bot$	`{}`
atomic concept	A	`A`
concept negation	$\neg\hat{C}$	`T - Ĉ`
conjunction	$\hat{C} \sqcap \hat{D}$	`Ĉ ∩ D`
disjunction	$\hat{C} \sqcup \hat{D}$	`Ĉ ∪ D`
existential restriction	$\exists R.\hat{C}$	`{i | i in T where exists(R[i] ∩ Ĉ)}` or `{i | i in T where card(R[i] ∩ Ĉ)>=1}`
universal restriction	$\forall R.\hat{C}$	`{i | i in T where R[i] ⊆ Ĉ)}`
Axioms	**Syntax**	**MiniZinc Syntax**
concept inclusion (subsumption)	$\hat{C} \sqsubseteq \hat{D}$	`Ĉ ⊆ D`
concept equivalence	$\hat{C} \equiv \hat{D}$	`Ĉ = D`

directly or indirectly in terms of themselves through axioms in a TBox $\mathcal{T}$ (Baader & Nutt, 2003), hence blocking is not necessary. Next, let us firstly define MiniZinc encoding for $\mathcal{ALC}$ and its sub-logics (Definition 5.1.1) formally.

Definition 5.1.1. $\mathcal{ALC}$2MiniZinc (MiniZinc Encoding : Set-based Encoding) Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$ TBox in normal form. Before we describe the encoding scheme, we will introduce some notations:

- `T` is a set of all individuals of $\mathcal{T}$ ($\Delta^{\mathcal{I}}$)
- `A` is a set of individuals of a concept A
- `R` is an array of set of individuals that are related to R
- A and B are atomic concepts
- i is an individual
- $\varphi^{\mathcal{T}}$ is a set of constraints

The encoding scheme is defined as follows:

ALC-ER1 For every axiom $\hat{A} \sqsubseteq \hat{B}$ ($i.e., \sqcap_n A_n \sqsubseteq \sqcup_m B_m$) $\in \mathcal{T}$,

$$((\bigcap_n \mathtt{A}_n) \subseteq (\bigcup_m \mathtt{B}_m)) \in \varphi^{\mathcal{T}} \tag{5.1.1}$$

ALC-ER2 For every axiom $A \sqsubseteq \exists R.B \in \mathcal{T}$,

$$\mathtt{A} \subseteq \{i \mid \forall i \in \mathtt{T} \wedge \mathtt{card(R[}i\mathtt{]} \cap \mathtt{B)} \geq 1\} \in \varphi^{\mathcal{T}} \tag{5.1.2}$$

ALC-ER3 For every axiom $\exists R.A \sqsubseteq B \in \mathcal{T}$,

$$\{i \mid \forall i \in \mathtt{T} \wedge \mathtt{card(R[}i\mathtt{]} \cap \mathtt{A)} \geq 1\} \subseteq \mathtt{B} \in \varphi^{\mathcal{T}} \tag{5.1.3}$$

ALC-ER4 For every axiom $A \sqsubseteq \forall R.B \in \mathcal{T}$,

$$\mathtt{A} \subseteq \{i \mid \forall i \in \mathtt{T} \wedge \mathtt{R[}i\mathtt{]} \subseteq \mathtt{B}\} \in \varphi^{\mathcal{T}} \tag{5.1.4}$$

ALC-ER5 For every axiom $\forall R.A \sqsubseteq B \in \mathcal{T}$,

$$\{i \mid \forall i \in \mathtt{T} \wedge \mathtt{R[}i\mathtt{]} \subseteq \mathtt{A}\} \subseteq \mathtt{B} \in \varphi^{\mathcal{T}} \tag{5.1.5}$$

ALC-ER6 For every concept name A for which we want to check satisfiability,

$$(\mathtt{A} \neq \emptyset) \in \varphi^{\mathcal{T}} \tag{5.1.6}$$

Note that the negative concept name $(\neg A)$ can be translated to (`T - A`), where ”`-`” is set difference.

Next, we present the intuition of our encoding. This encoding is straightforward regarding the semantics of $\mathcal{ALC}$ in Table 2.5. We assume that an acyclic $\mathcal{ALC}$ TBox $\mathcal{T}$ in normal form as described in Section 2.4.2. $\mathcal{T}$ can be encoded by applying the encoding rules in Definition 5.1.1. Let i be an individual, n, m be non-negative integers, `T` be a set of all individuals in the domain and $\varphi^{\mathcal{T}}$ be a set of MiniZinc constraints. Axioms in $\mathcal{T}$ are encoded as follows, and added to $\varphi^{\mathcal{T}}$:

- For every axiom $\hat{A} \sqsubseteq \hat{B}$ $(i.e., \sqcap_n A_n \sqsubseteq \sqcup_m B_m)$, the **ALC-ER1** rule is applied on this axiom to obtain the constraint of type (5.1.1). This rule is straightforward since the conjunction and disjunction can be easily translated to the intersect operation of sets $\mathtt{A}_i$ and the union operation of sets $\mathtt{B}_i$ respectively, and the subsumption $\sqsubseteq$ can be translated into the subset operation in MiniZinc.

- For every axiom $A \sqsubseteq \exists R.B$, the **ALC-ER2** rule is applied on this axiom to obtain the constraint of type (5.1.2). The subsumption $\sqsubseteq$ is encoded into the subset operation. This constraint means that `A` is subset of a set of individuals i such that $\forall i \in \Delta^{\mathcal{I}}$ and the cardinality of the intersection of a set of R-successors of i and `B` is greater than 1. As can be seen, the meaning of this constraint is very close to the semantics of $\exists R.B$ in Table 2.5.

- For every axiom $\exists R.A \sqsubseteq B$, the **ALC-ER3** rule is applied on this axiom to obtain the constraint of type (5.1.3). This constraint means that a set of individuals i such that $\forall i \in \Delta^{\mathcal{I}}$ and the cardinality of the intersection of a set of R-successors of i and `A` is greater than 1 is a subset of `B`.

- For every axiom $A \sqsubseteq \forall R.B$, the **ALC-ER4** rule is applied on this axiom to obtain the constraint of type (5.1.4). This constraint means that `A` is subset of a set of individuals i such that $\forall i \in \Delta^{\mathcal{I}}$ and a set of all R-successors of i is a subset of `B`. As can be seen, the meaning of this constraint is very close to the semantics of $\forall R.B$ in Table 2.5.

- For every axiom $\forall R.A \sqsubseteq B$, the **ALC-ER5** rule is applied on this axiom to obtain the constraint of type (5.1.5). This constraint means that a set of individuals i, such that $\forall i \in \Delta^{\mathcal{I}}$ and a set of all R-successors of i is a subset of `A`, is a subset of `B`.

- To check the satisfiability of concept A, a constraint $\mathtt{A} \neq \emptyset$ is added (Rule **ALC-ER6**). This constraint means that a set `A` is not empty. If this constraint is satisfiable w.r.t. all constraints in $\varphi^{\mathcal{T}}$, then the concept A is satisfiable.

- To check subsumption of $A \sqsubseteq B$, this can be done by adding a constraint `A` $\cap$ `(T - B)` $\neq \emptyset$. If this constraint is unsatisfiable w.r.t. all constraints in $\varphi^{\mathcal{T}}$, then $\mathcal{T} \models A \sqsubseteq B$.

As can be seen, the syntax of MiniZinc is quite similar to the syntax of $\mathcal{ALC}$. In addition, it is easy to encode ontologies in $\mathcal{ALC}$ to MiniZinc models.

5.1.2 Finiteness of MiniZinc Models

Since inference in CP requires a finite and known size of the universe, we need to calculate the number of individuals needed for the MiniZinc model. This number needs to be big enough to guarantee the completeness of our CP-based reasoning approach, i.e., if the solver returns *unsatisfiable* for a MiniZinc model, the corresponding ontology is indeed inconsistent.

The number of necessary individuals to satisfy all axioms in a TBox $\mathcal{T}$ in normal form is calculated by the following rules. Let us represent an individual uniquely by its label σ. A label σ can be a non-zero number or $\sigma'.R.n$, where σ' denotes another label, R denotes a role, and $n \geq 1$. Let $\beta^{\mathcal{I}}$ be a set of individuals and Δ_C be a set of individuals of concept C.

INV1 Initialisation: $\{1\} \subset \beta^{\mathcal{I}}$ and $\{1\} \subset \Delta_{C_i}$ where $\sqcap_i C_i \sqsubseteq \hat{D} \in \mathcal{T}$

INV2 If $\sigma \in \beta^{\mathcal{I}}$, $\sqcap_i C_i \sqsubseteq \sqcup_j D_j \in \mathcal{T}$, $\sigma \in \Delta_{C_i}$, then

$$\{\sigma\} \subset \Delta_{D_j}$$

INV3 If $\sigma \in \beta^{\mathcal{I}}$, $\sigma \in \Delta_C$, and $C \sqsubseteq \exists R.D \in \mathcal{T}$, then

$$\{\sigma.R.i\} \subset \beta^{\mathcal{I}}$$
$$\{\sigma.R.i\} \subset \Delta_D$$

INV4 If $\sigma \in \beta^{\mathcal{I}}$, $\sigma \in \Delta_{\neg C}$, and $\forall R.D \sqsubseteq C \in \mathcal{T}$, then

$$\{\sigma.R.i\} \subset \beta^{\mathcal{I}}$$
$$\{\sigma.R.i\} \subset \Delta_{\neg D}$$

INV5 If $\sigma \in \beta^{\mathcal{I}}$, $\sigma \in \Delta_C$, and $C \sqsubseteq \forall R.D \in \mathcal{T}$, then

$$\{\sigma.R.i \mid \sigma.R.i \in \beta^{\mathcal{I}}\} \subset \Delta_D$$

INV6 If $\sigma \in \beta^{\mathcal{I}}$, $\sigma \in \Delta_{\neg C}$, and $\exists R.D \sqsubseteq C \in \mathcal{T}$, then

$$\{\sigma.R.i \mid \sigma.R.i \in \beta^{\mathcal{I}}\} \subset \Delta_{\neg D}$$

These rules are applied exhaustively on $\mathcal{T}$ until no more rules can be applied. The rules are triggered with respect to individuals assigned to the left-hand side of axioms. Since we only consider acyclic TBoxes, the application of rules terminates. Finally, the number of required individuals is the size of $\beta^{\mathcal{I}}$.

Note that this algorithm only calculates the number of required individuals for MiniZinc models. In other words, we are only interested in the size of $\beta^{\mathcal{I}}$, but not the individuals assigned to concepts. This algorithm gives a safe bound for soundness and completeness of the reasoning approach, i.e., if $\mathcal{T}$ has a model, the bound calculated will be larger than the size of the smallest model.

Intuitively, the number of individuals is determined with respect to the structure of TBox. The initialisation rule **INV1** assigns the root node 1 to the left-hand side of every axiom in $\mathcal{T}$. Rule **INV2** handles the axiom $\sqcap_i C_i \sqsubseteq \sqcup_j D_j$. If an individual σ is in Δ_{C_i}, it is added to Δ_{D_j}. Rule **INV3** handles the axiom $C \sqsubseteq \exists R.D$. If an individual σ is in Δ_C, it is added to $\Delta_{\exists R.D}$ and then a new individual $\sigma.R.i$, which is a member of Δ_D, is generated to satisfy R. In addition, the new individual $\sigma.R.i$ is added to $\beta^{\mathcal{I}}$. Rule **INV4** handles the axiom $\forall R.D \sqsubseteq C$. This axiom can be transformed to $\neg C \sqsubseteq \exists R.\neg D$. The idea of this rule is similar to **INV3**. If an individual σ is in $\Delta_{\neg C}$, it is added to $\Delta_{\exists R.\neg D}$ and then a new individual $\sigma.R.i$, which is a member of $\Delta_{\neg D}$, is generated to satisfy R. In addition, the new individual $\sigma.R.i$ is added to $\beta^{\mathcal{I}}$. Rule **INV5** handles the axiom $C \sqsubseteq \forall R.D$. If an individual σ is in Δ_C, all its R-successors need to be added to Δ_D. Rule **INV6** handles the axiom $\exists R.D \sqsubseteq C$. This axiom can be transformed to $\neg C \sqsubseteq \forall R.\neg D$. The idea of this rule is similar to **INV5**. If an individual σ is in $\Delta_{\neg C}$, it is added to $\Delta_{\forall R.\neg D}$ and then all R-successors need to be added to $\Delta_{\neg D}$.

The number of individuals calculated by this algorithm is correct since the application of these rules is similar to that of the tableau algorithm as described above. Individuals are generated to satisfy existential restrictions by rules **INV3** and **INV4**. When the tableau algorithm generates individuals, this algorithm generates individuals as well. This is the reason why the algorithm calculates the bound universe as big as the model generated by the tableau algorithm. Note that the number of individuals calculated by this algorithm may not be the smallest number, hence may not be optimal. As can be easily seen, the number of individuals is exponential to the nesting depth of restrictions.

Let us use Example 5.1.1 to illustrate the application of these rules.

Example 5.1.1. Given a TBox $\mathcal{T}$ containing an axiom $C \sqsubseteq \exists R.A \sqcap \exists R.B \sqcap \forall R.(\exists R.A \sqcap \exists R.B)$, we start by normalising $\mathcal{T}$ to $\mathcal{T}'$ as follows:

$$\begin{array}{ll} C \sqsubseteq \exists R.A & C \sqsubseteq \exists R.B \\ C \sqsubseteq \forall R.D & D \sqsubseteq \exists R.A \\ D \sqsubseteq \exists R.B & \end{array}$$

where D is a new concept name.

Firstly, we apply **INV1** to $\mathcal{T}'$. An individual 1, is added to C, D, and $\Delta^{\mathcal{I}}$. Then we apply **INV2** to $C \sqsubseteq \exists R.A$ for each individual in C. A new individual $1.R.1$ will be generated and added to A and $\Delta^{\mathcal{I}}$. Next, we apply **INV2** to $C \sqsubseteq \exists R.B$ for each individual in C. A new individual $1.R.2$ will be generated and added to B and $\Delta^{\mathcal{I}}$. Now, $\Delta^{\mathcal{I}}$ contains three individuals $1, 1.R.1, 1.R.2$. Then we apply **INV5** to $C \sqsubseteq \forall R.D$ for each individual in C. The individuals $1.R.1$ and $1.R.2$ will be added to D. Now, D contains individuals, $1, 1.R.1, 1.R.2$. Next, we apply **INV3** to $D \sqsubseteq \exists R.A$ for each individual in D. This will generate three more individuals $1.R.3, 1.R.1.R.1, 1.R.2.R.1$, added to A and $\Delta^{\mathcal{I}}$. Then we apply **INV3** to $D \sqsubseteq \exists R.B$ for each individual in D. This will generate three more individuals $1.R.4, 1.R.1.R.2, 1.R.2.R.2$, added to B and $\Delta^{\mathcal{I}}$. Finally, there is no more rule to be applied and $\Delta^{\mathcal{I}}$ contains individuals $1, 1.R.1, 1.R.2, 1.R.3, 1.R.1.R.1, 1.R.2.R.1, 1.R.4, 1.R.1.R.2, 1.R.2.R.2$. As a result, the number of individuals is 9. □

As can be seen, an $\mathcal{ALC}$ ontology can be encoded into a MiniZinc model in linear time. However, there is a blow up in the size of the encoding due to the number of individuals required for a MiniZinc model, when the MiniZinc model is flattened into FlatZinc for solving. FlatZinc is a solver input language that is supported by a wide range of solvers (Becket, 2014).

Regarding this procedure, the process of calculating the number of individuals will be an overhead of our approach, especially in the case of an ontology that requires a large number of individuals. The pre-calculation of the number of individuals is also where we may add blocking techniques to our approach. The following is a sketch how such a blocking approach would keep track of axioms for which the algorithm already added new individuals to the domain of superclasses, as well as of superclasses and subclasses of axioms. When a new individual is added to the domain of superclasses, it will trigger the pre-calculation rules from above, where superclasses become subclasses of other axioms. If the subclasses and axioms have already been considered by the pre-calculation before, the process can be blocked. Since we only focus on acyclic TBoxes, we leave the full discussion and implementation of these ideas for future work.

5.1.3 Encoding Rules Implementation

In this section, we present the implementation of the encoding rules defined in Definition 5.1.1. We will show how can we translate each constraint into the actual MiniZinc syntax. Some of them can be mapped directly the actual MiniZinc syntax. For example, $\subseteq$ can be translated to a `subset` operation or an implication (`->`). For equivalence ($\equiv$), it can be translated into either `=` or `<->`. The implementation is presented as follows:

- For the rule **ALC-ER1**, the constraint $(\bigcap_n \mathtt{A}_n) \subseteq (\bigcup_m \mathtt{B}_m)$ of type (5.1.1) is implemented as

  constraint ($\mathtt{A}_1$ **intersect** ... $\mathtt{A}_n$ **subset** $\mathtt{B}_1$ **union** ... $\mathtt{B}_m$) (5.1.3.1)

  $\subseteq$ is implemented as the `subset` operation. The conjunctions $\cap$ and disjunctions $\cup$ are implemented as the `intersect` and `union` operations respectively. $\mathtt{A}_i$ and $\mathtt{B}_i$ are implemented as set variables.

- For the rule **ALC-ER2**, the constraint $\mathtt{A} \subseteq \{i \mid \forall i \in \mathtt{T} \wedge \mathtt{card(R[}i\mathtt{]} \cap \mathtt{B)} \geq 1\}$ of type (5.1.2) is implemented as

  ```
  constraint forall (i in A) (card(R[i] intersect B) >= 1) (5.1.3.2)
  ```

  For efficiency, $\subseteq$ is implemented as an implication implicitly. This constraint means that if an individual `i` is in `A`, then the cardinality of the intersection of R-successors of `i` and `B` is greater than 1.

- For the rule **ALC-ER3**, the constraint $\{i \mid \forall i \in \mathtt{T} \wedge \mathtt{card(R[}i\mathtt{]} \cap \mathtt{A)} \geq 1\} \subseteq \mathtt{B}$ of type (5.1.3) is implemented as

  ```
  constraint forall (i in T)
          ((card(R[i] intersect A) >= 1) -> i in B)        (5.1.3.3)
  ```

  In this case, $\subseteq$ is implemented as an implication `->` explicitly. This constraint means that for all individuals `i` in the domain, if the cardinality of the intersection of R-successors of `i` and `A` is greater than 1, then the individual `i` is in `B`.

- For the rule **ALC-ER4**, the constraint $\mathtt{A} \subseteq \{i \mid \forall i \in \mathtt{T} \wedge \mathtt{card(R[}i\mathtt{]} \subseteq \mathtt{B}\}$ of type (5.1.4) is implemented as

  ```
  constraint forall (i in A) (R[i] subset B)                (5.1.3.4)
  ```

  Similar to the rule **ALC-ER2**, the first $\subseteq$ is implemented as an implication implicitly. This constraint means that if an individual `i` is in `A`, then the R-successors of `i` is a subset of `B`.

- For the rule **ALC-ER5**, the constraint $\{i \mid \forall i \in \mathtt{T} \wedge \mathtt{card(R[}i\mathtt{]} \subseteq \mathtt{A}\} \subseteq \mathtt{B}$ of type (5.1.5) is implemented as

```
constraint forall (i in T)
        ((R[i] subset A) -> i in B)                    (5.1.3.5)
```

This constraint means that for all individuals `i` in the domain, if the R-successors of `i` is a subset of `A`, then an individual `i` are in `B`.

- For the rule **ALC-ER6**, the constraint $\mathtt{A} \neq \emptyset$ of type (5.1.6) is implemented as `card(A) != 0`, which means that the cardinality of `A` is not equal to zero.

5.2 Correctness and Completeness of the Encoding

In this section, we conduct proofs to prove the soundness and completeness of our encoding approach. The proof proceeds by model construction, which yields an interpretation from a given MiniZinc solution (for soundness) as well as the reverse direction (for completeness). We are going to ignore the fact that we need to know the size of domain in the MiniZinc encoding, assuming that we know the size of domain, which may be potentially large.

Theorem 5.2.1. *Given an acyclic $\mathcal{ALC}$ TBox $\mathcal{T}$ in normal form and the MiniZinc encoding, the concept A is satisfiable w.r.t. $\mathcal{T}$ if and only if the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (S_A \neq \emptyset)$ is satisfiable.*

Proof. It is a direct consequence of the following Lemmas. □

Lemma 5.2.2. *(Soundness) Given an acyclic $\mathcal{ALC}$ TBox $\mathcal{T}$ in normal form and the MiniZinc encoding, if the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable then the concept A is satisfiable w.r.t. $\mathcal{T}$.*

The main proof idea is to construct an interpretation $\mathcal{I}$ from a solution of the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$, and then show by a straightforward inductive argument that $\mathcal{I}$ is a model for $\mathcal{T}$ and has a non-empty concept A.

Proof. Let $\varphi^{\mathcal{T}}$ be the MiniZinc formula of $\mathcal{T}$. $\mathcal{S} = (\mathtt{T}^S, S_C, S_R)$ is a tuple of assignments to variables in the MiniZinc formula, where $\mathtt{T}^S$ represents the set of individuals appearing in the encoding, S_C represents a set of assignments of concept name variables $\mathtt{A}^S$, S_R represents a set of assignments of role variables $\mathtt{R}^S$ and $\mathcal{S}$ is a solution of $\varphi^{\mathcal{T}}$. $\Delta^{\mathcal{I}}$ denotes the domain of $\mathcal{I}$. Let a, b be individuals. We must prove that there also exists a model for $\mathcal{T}$. From $\mathcal{S}$, we define an interpretation $\mathcal{I}$ as follows:

$$\Delta^{\mathcal{I}} = \mathtt{T}^S, \tag{5.2.1}$$

$$A^{\mathcal{I}} = \mathtt{A}^S, \tag{5.2.2}$$

$$R^{\mathcal{I}} = \{(a, b) \mid a, b \in \Delta^{\mathcal{I}} \text{ and } b \in \mathtt{R}^S\mathtt{[}a\mathtt{]}\}, \tag{5.2.3}$$

For non-name concepts, we define $\mathcal{I}$ such that:

$$(\sqcap_i A_i)^{\mathcal{I}} = \{a \mid a \in \Delta^{\mathcal{I}} \text{ and } a \in (\bigcap_i \mathtt{A}_i)^S\} \tag{5.2.4}$$

$$(\sqcup_j B_j)^{\mathcal{I}} = \{a \mid a \in \Delta^{\mathcal{I}} \text{ and } a \in (\bigcup_j \mathtt{B}_j)^S\} \tag{5.2.5}$$

We prove by induction over the structure of $\mathcal{T}$ that $\mathcal{I}$ is semantically consistent and it is a model of $\mathcal{T}$. For this purpose, for every axiom $\hat{C} \sqsubseteq \hat{D} \in \mathcal{T}$ in normal form and every individual a, we must prove that $\mathcal{I}$ satisfies the following condition: if $a \in \hat{C}^{\mathcal{I}}$ then $a \in \hat{D}^{\mathcal{I}}$ (i.e. $\hat{C}^{\mathcal{I}} \subseteq \hat{D}^{\mathcal{I}}$, respecting the semantics of the axiom $\hat{C} \sqsubseteq \hat{D}$). When we talk about the semantic of concepts, we always refer to Table 2.5.

The condition trivially follows from point 1-5 of Definition 5.1.1. Let us consider the following cases of axioms: (1) $A \sqsubseteq B$, (2) $\sqcap_i A_i \sqsubseteq B$, (3) $A \sqsubseteq \sqcup_j B_j$, (4) $A \sqsubseteq \exists R.B$, (5) $\exists R.A \sqsubseteq B$, (6) $A \sqsubseteq \forall R.B$, and (7) $\forall R.A \sqsubseteq B$. Any axiom of $\mathcal{T}$ in normal form is a sub-case of one among (1)-(7).

(1) By hypothesis, we have $a \in A^{\mathcal{I}}$. Thus, $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (5.2.2). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ALC-ER1**) $\varphi^{\mathcal{T}}$ contains the constraint $(\mathtt{A} \subseteq \mathtt{B})$ of type (5.1.1). It follows $a \in \mathtt{B}^S$ since $\varphi^{\mathcal{T}}$ is satisfiable. Therefore, $a \in B^{\mathcal{I}}$ by (5.2.2).

(2) By hypothesis, we have $a \in (\sqcap_i A_i)^{\mathcal{I}}$. Then $a \in (\bigcap_i \mathtt{A}_i)^S$ by definition of $\mathcal{I}$ (5.2.4). Since $\bigcap_i \mathtt{A}_i$ is in $\varphi^{\mathcal{T}}$, (**ALC-ER1**) $\varphi^{\mathcal{T}}$ contains the constraint $((\bigcap_i \mathtt{A}_i) \subseteq \mathtt{B})$ of type (5.1.1). It follows that $a \in \mathtt{B}^S$ because $\varphi^{\mathcal{T}}$ is satisfiable. Thus, $a \in B^{\mathcal{I}}$ by definition of $\mathcal{I}$ (5.2.2).

(3) By hypothesis, we have $a \in A^{\mathcal{I}}$. Then $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (5.2.2). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ALC-ER1**) $\varphi^{\mathcal{T}}$ contains the constraint $(\mathtt{A} \subseteq (\bigcap_i \mathtt{B}_j))$ of type (5.1.1). It follows that $a \in (\bigcap_i \mathtt{B}_j)^S$ because $\varphi^{\mathcal{T}}$ is satisfiable. Thus, $a \in (\sqcup_j B_j)^{\mathcal{I}}$ by definition of $\mathcal{I}$ (5.2.5).

(4) By hypothesis, we have $a \in A^{\mathcal{I}}$. Therefore, $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (5.2.2). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ALC-ER2**) $\varphi^{\mathcal{T}}$ contains the constraint $\mathtt{A} \subseteq \{a \mid \forall a \in \mathtt{T} \wedge \mathtt{card(R[}a\mathtt{]} \cap \mathtt{B)} \geq 1\}$ of type (5.1.2). Due to this constraint, $\varphi^{\mathcal{T}}$ can be satisfiable only if $a \in \{a \mid \forall a \in \mathtt{T} \wedge \mathtt{card(R[}a\mathtt{]} \cap \mathtt{B)} \geq 1\}$ is true. Thus, there exists at least one individual b such that $b \in \mathtt{T}^S$, $b \in \mathtt{R}^S\mathtt{[}a\mathtt{]}$ and $b \in \mathtt{B}^S$ because $\varphi^{\mathcal{T}}$ is satisfiable. By definitions of $\mathcal{I}$ (5.2.3) and (5.2.2), $(a, b) \in R^{\mathcal{I}}$ and $b \in B^{\mathcal{I}}$. As a result, $a \in (\exists R.B)^{\mathcal{I}}$.

(5) By hypothesis, we have $a \in (\exists R.A)^{\mathcal{I}}$. Therefore, there exists at least one individual b such that $b \in \mathtt{T}^S$, $b \in \mathtt{R}^S\mathtt{[}a\mathtt{]}$ and $b \in \mathtt{A}^S$ by definitions of $\mathcal{I}$ (5.2.3) and (5.2.2). Since $\{a \mid \forall a \in \mathtt{T} \wedge \mathtt{card(R[}a\mathtt{]} \cap \mathtt{A)} \geq 1\}$ is in $\varphi^{\mathcal{T}}$, (**ALC-ER3**) $\varphi^{\mathcal{T}}$ contains the constraint $\{a \mid \forall a \in \mathtt{T} \wedge \mathtt{card(R[}a\mathtt{]} \cap \mathtt{A)} \geq 1\} \subseteq \mathtt{B}$ of type (5.1.3). It follows $a \in \mathtt{B}^S$ since $\varphi^{\mathcal{T}}$ is satisfiable. Therefore, $a \in B^{\mathcal{I}}$ by (5.2.2).

(6) By hypothesis, we have $a \in A^{\mathcal{I}}$. Therefore, $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (5.2.2). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ALC-ER4**) $\varphi^{\mathcal{T}}$ contains the constraint $\mathtt{A} \subseteq \{a \mid \forall a \in \mathtt{T} \wedge \mathtt{R[}a\mathtt{]} \subseteq \mathtt{B}\}$ of type (5.1.4). Due to this constraint, $\varphi^{\mathcal{T}}$ can be satisfiable only if $a \in \{a \mid \forall a \in \mathtt{T} \wedge \mathtt{R[}a\mathtt{]} \subseteq \mathtt{B}\}$ is true. Thus, all individuals b, such that $b \in \mathtt{T}^S$ and $b \in \mathtt{R}^S\mathtt{[}a\mathtt{]}$, are subset of $\mathtt{B}^S$ because $\varphi^{\mathcal{T}}$ is satisfiable. By definitions of $\mathcal{I}$ (5.2.3) and (5.2.2), $(a, b) \in R^{\mathcal{I}}$ and $b \in B^{\mathcal{I}}$. As a result, $a \in (\forall R.B)^{\mathcal{I}}$.

(7) By hypothesis, we have $a \in (\forall R.A)^{\mathcal{I}}$. Therefore, all individuals b, such that $b \in \mathtt{T}^S$ and $b \in \mathtt{R}^S\mathtt{[}a\mathtt{]}$, are subset of $\mathtt{A}^S$ by definitions of $\mathcal{I}$ (5.2.3) and (5.2.2). Since $\{a \mid \forall a \in \mathtt{T} \wedge \mathtt{R[}a\mathtt{]} \subseteq \mathtt{A}\}$ is in $\varphi^{\mathcal{T}}$, (**ALC-ER6**) $\varphi^{\mathcal{T}}$ contains the constraint $\{a \mid \forall a \in \mathtt{T} \wedge \mathtt{R[}a\mathtt{]} \subseteq \mathtt{A}\} \subseteq \mathtt{B}$ of type (5.1.5). It follows $a \in \mathtt{B}^S$ since $\varphi^{\mathcal{T}}$ is satisfiable. Therefore, $a \in B^{\mathcal{I}}$ by (5.2.2).

For $\neg B$, by hypothesis, let $a \in (\neg B)^{\mathcal{I}}$. By definition of (5.2.2), $a \in \Delta^{\mathcal{I}}$ and $a \in \mathtt{T}^S - \mathtt{B}^S$. It follows $a \notin B^{\mathcal{I}}$. Thus, $a \in \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$.

Finally, we prove that if $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable, then there exists an interpretation $\mathcal{I}$, such that $\mathcal{I}$ is a model for $\mathcal{T}$ and $A^{\mathcal{I}} \neq \emptyset$. $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable if and only if $\varphi^{\mathcal{T}}$ is satisfiable. Since $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable, there exists an individual a such that $a \in \mathtt{A}^S$. By definition (5.2.2), we have $a \in A^{\mathcal{I}}$. Thus, the concept A is satisfiable w.r.t. $\mathcal{T}$. □

Lemma 5.2.3. *(Completeness) Given an acyclic $\mathcal{ALC}$ TBox $\mathcal{T}$ in normal form and the MiniZinc encoding, if the concept A is satisfiable w.r.t. $\mathcal{T}$ then the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable.*

The main idea of this proof is to construct a solution $\mathcal{S}$ from a model $\mathcal{I}$ of $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$, and then show that $\mathcal{S}$ is a solution of $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$. We assume that a model $\mathcal{I}$ is constructed by the completion rules in (Baader & Nutt, 2003).

Proof. Given that the concept A is satisfiable w.r.t. $\mathcal{T}$, there exists a model $\mathcal{I}$ for $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$. The completion rules in (Baader & Nutt, 2003) are sound, complete and terminating. Therefore, we can construct a model $\mathcal{I}$ for $\mathcal{T}$, and from $\mathcal{I}$ build a solution $\mathcal{S}$ for $\varphi^{\mathcal{T}}$.

$\mathcal{S} = (\mathtt{T}^S, S_C, S_R)$ is a solution tuple of a MiniZinc formula $\varphi^{\mathcal{T}}$ as introduced above. $\Delta^{\mathcal{I}}$ denotes the domain of $\mathcal{I}$. Let a, b be individuals. From $\mathcal{I}$, we can define the solution $\mathcal{S}$ as follows:

$$\mathtt{T}^S = \Delta^{\mathcal{I}} \tag{5.2.6}$$

$$\mathtt{A}^S = A^{\mathcal{I}} \tag{5.2.7}$$

$$\mathtt{R}^S\mathtt{[}a\mathtt{]} = \{b \mid a, b \in \mathtt{T}^S \text{ and } (a,b) \in R^{\mathcal{I}}\} \tag{5.2.8}$$

Now, we must prove that $\mathcal{S}$ satisfies all the constraints of $\varphi^{\mathcal{T}}$ such that $a \in \Delta^{\mathcal{I}}$ for every type of constraint from (5.1.1) to (5.1.5).

Constraints of type (5.1.1) represent the MiniZinc encoding of an axiom $\hat{A} \sqsubseteq \hat{B}$ ($\sqcap_n A_n \sqsubseteq \sqcup_m B_m$). We can distinguish two cases:

1. An axiom $A_1 \sqcap A_2 \sqsubseteq B$, where A_1, A_2 and B are basic concepts, is encoded to $\mathtt{A}_1 \cap \mathtt{A}_2 \subseteq \mathtt{B}$. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $(A_1 \sqcap A_2)^{\mathcal{I}} \subseteq B^{\mathcal{I}}$. Thus, if $a \in A_1^{\mathcal{I}} \cap A_2^{\mathcal{I}}$, then $a \in B^{\mathcal{I}}$. This follows by 5.2.7 that $a \in \mathtt{A}_1^S$, $a \in \mathtt{A}_1^S$ and $a \in \mathtt{B}^S$. Hence, the constraint is satisfied.

2. An axiom $A \sqsubseteq B_1 \sqcup B_2$, where A, B_1 and B_2 are basic concepts, is encoded to $\mathtt{A} \subseteq \mathtt{B}_1 \cup \mathtt{B}_2$. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $A^{\mathcal{I}} \subseteq (B_1 \sqcup B_2)^{\mathcal{I}}$. Thus, if $a \in A^{\mathcal{I}}$, then $a \in B_1^{\mathcal{I}} \cup B_2^{\mathcal{I}}$. This follows by 5.2.7 that $a \in \mathtt{A}^S$ and either $a \in \mathtt{B}_1^S$ or $a \in \mathtt{B}_2^S$. Hence, the constraint is satisfied.

Constraints of type (5.1.2) represent the MiniZinc encoding of an axiom $A \sqsubseteq \exists R.B$. We must show that the constraint $\mathtt{A} \subseteq \{a \mid \forall a \in \mathtt{T} \wedge \mathbf{card}\mathtt{(R[}a\mathtt{]} \cap \mathtt{B)} \geq 1\}$ of type (5.1.2) is satisfied. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $A^{\mathcal{I}} \subseteq (\exists R.B)^{\mathcal{I}}$. Thus, if $a \in A^{\mathcal{I}}$, then $a \in (\exists R.B)^{\mathcal{I}}$. Since $a \in (\exists R.B)^{\mathcal{I}}$, $(a,b) \in R^{\mathcal{I}}$ and $b \in B^{\mathcal{I}}$ such that $b \in \Delta^{\mathcal{I}}$. By definition of $\mathcal{S}$ (5.2.7) and (5.2.8), we have $a \in \mathtt{A}^S$, $b \in \mathtt{R}^S\mathtt{[}a\mathtt{]}$, and $b \in \mathtt{B}^S$. As a result, the constraint is satisfied.

Constraints of type (5.1.3) represent the MiniZinc encoding of an axiom $\exists R.A \sqsubseteq B$. We must show that the constraint $\{a \mid \forall a \in \mathtt{T} \wedge \mathbf{card}\mathtt{(R[}a\mathtt{]} \cap \mathtt{A)} \geq 1\} \subseteq \mathtt{B}$ of type (5.1.2) is satisfied. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $(\exists R.A)^{\mathcal{I}} \subseteq B^{\mathcal{I}}$. Thus, if $a \in (\exists R.A)^{\mathcal{I}}$, then $a \in B^{\mathcal{I}}$. Since $a \in (\exists R.A)^{\mathcal{I}}$, $(a,b) \in R^{\mathcal{I}}$ and $b \in A^{\mathcal{I}}$ such that $b \in \Delta^{\mathcal{I}}$.

By definition of $\mathcal{S}$ (5.2.7) and (5.2.8), we have $b \in \texttt{R}^S\texttt{[}a\texttt{]}$, $b \in \texttt{A}^S$, and $a \in \texttt{B}^S$. Hence, the constraint is satisfied.

Constraints of type (5.1.4) represent the MiniZinc encoding of an axiom $A \sqsubseteq \forall R.B$. We must show that the constraint $\texttt{A} \subseteq \{a \mid \forall a \in \texttt{T} \land \textbf{card}\texttt{(R[}a\texttt{]} \subseteq \texttt{B}\}$ of type (5.1.4) is satisfied. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $A^{\mathcal{I}} \subseteq (\forall R.B)^{\mathcal{I}}$. Thus, if $a \in A^{\mathcal{I}}$, then $a \in (\forall R.B)^{\mathcal{I}}$. Since $a \in (\forall R.B)^{\mathcal{I}}$, for all $(a,b) \in R^{\mathcal{I}}$, $b \in B^{\mathcal{I}}$ such that $b \in \Delta^{\mathcal{I}}$. By definition of $\mathcal{S}$ (5.2.7) and (5.2.8), we have $a \in \texttt{A}^S$, $b \in \texttt{R}^S\texttt{[}a\texttt{]}$, and $b \in \texttt{B}^S$. As a result, the constraint is satisfied.

Constraints of type (5.1.5) represent the MiniZinc encoding of an axiom $\forall R.A \sqsubseteq B$. We must show that the constraint $\{a \mid \forall a \in \texttt{T} \land \textbf{card}\texttt{(R[}a\texttt{]} \subseteq \texttt{A}\} \subseteq \texttt{B}$ of type (5.1.5) is satisfied. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $(\forall R.A)^{\mathcal{I}} \subseteq B^{\mathcal{I}}$. Thus, if $a \in (\forall R.A)^{\mathcal{I}}$, then $a \in B^{\mathcal{I}}$. Since $a \in (\forall R.A)^{\mathcal{I}}$, for all $(a,b) \in R^{\mathcal{I}}$, $b \in A^{\mathcal{I}}$ such that $b \in \Delta^{\mathcal{I}}$. By definition of $\mathcal{S}$ (5.2.7) and (5.2.8), we have $b \in \texttt{R}^S\texttt{[}a\texttt{]}$, $b \in \texttt{A}^S$, and $a \in \texttt{B}^S$. Hence, the constraint is satisfied.

For $\neg A$, let $a \in (\neg A)^{\mathcal{I}}$. By definition of (5.2.7), $a \in S_{\mathcal{T}}$ and $a \in \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$. It follows $a \notin A^{\mathcal{I}}$. Thus, $a \in \texttt{T}^S - \texttt{B}^S$.

Finally, since there exists a model $\mathcal{I}$ for $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$, there exists an individual $a \in \Delta^{\mathcal{I}}$ such that $a \in A^{\mathcal{I}}$. By definition 5.2.7, we have $a \in \texttt{A}^S$. Hence, the MiniZinc formula $\varphi^{\mathcal{T}} \land (\texttt{A} \neq \emptyset)$ is satisfiable.

□

5.3 Empirical Evaluation

In order to verify empirically the effectiveness of our approach, we conducted an evaluation with about 500 ontologies from an extensive set of benchmarks. This evaluation is to show that our approach is competitive with tableau reasoners. For this evaluation, we performed *consistency checking* on $\mathcal{ALC}$ ontologies to evaluate our approach against two of the fastest reasoners, HermiT (Shearer et al., 2008; Glimm et al., 2014) and Konclude (Steigmiller et al., 2014), and the intelligent tableau reasoner, Light (Zuo & Haarslev, 2013). The reason for our focus on consistency checking is that all other related works mentioned focused on this task (Meissner, 2011; Haarslev, Sebastiani, & Vescovi, 2011; Zuo & Haarslev, 2013). We leave the other reasoning tasks as future work.

We have implemented the encoder `ALC2MiniZinc` in Java to encode an ontology into a MiniZinc model. In combination with `ALC2MiniZinc`, we have selected several CP solvers to use with our approach. We generated MiniZinc models using our `ALC2MiniZinc`, and then solved the models using `Opturion CPX (ocpx)`[1] (v. 1.0.2) and `chuffed`[2], which are lazy clause generation CP solvers, and `Gecode`[3] (v. 4.4.0). We compare the runtimes of our system with those of tableau-based reasoners HermiT[4] (v. 1.3.8), Konclude[5] (v. 0.6.2), and Light[6] (v. 0.2).

A runtime limit of 300 seconds was used for the first evaluation. We limited a runtime of 100 seconds for the second evaluation. All results presented in this section have been obtained on a 64bit quad-core `Intel Core i5 3.2GHz` machine, with `8GB` of RAM under

[1]`http://www.opturion.com/#!cpx/ch52y`
[2]`https://github.com/geoffchu/chuffed`
[3]`http://www.gecode.org/`
[4]`http://www.hermit-reasoner.com/`
[5]`http://www.derivo.de/en/products/konclude.html`
[6]`http://www.lightreasoner.co.nf/`

ubuntu 16.04. We used MiniZinc version 2.1.3[7]. Note that the results presented in this section include time taken by preprocessing including the pre-calculation of number of individuals and normalisation, encoding (taken by `ALC2MiniZinc`), flattening (translating form MiniZinc to FlatZinc), and solving. FlatZinc is a solver input language that is supported by a wide range of solvers (Becket, 2014).

5.3.1 Evaluation Description

We used three different sets of benchmarks to test our encoding. The first set is a set of acyclic $\mathcal{ALC}$ ontologies, which come from ORE2014 Reasoner Competition[8]. This benchmark data set is collected from the MOWLCorp (Manchester OWL Corpus), which was obtained through a Web Crawl, Google Custom Search API and user submissions, the Oxford Ontology Library, and a BioPortal Snapshot (June 2014). The ontologies in this data set have various structure. Note that the ORE2014 data set contains many ontologies that have different expressivity, but we have implemented a filter to retain only $\mathcal{ALC}$ ontologies from the dataset. The second set (JNH (Hoos, 2011)) was used as an extreme case to demonstrate poor performance of tableau-based reasoners (Zuo & Haarslev, 2013). It has been generated by converting SAT benchmarks (Hoos, 2011) into OWL syntax. This benchmark data set is used to test DL reasoners for dealing with ontologies, that contain a lot of disjunctions and demonstrates how constructs such as chains of disjunctions can lead to inefficiencies in tableau-based reasoners. The third set of benchmarks come from Tableaux'98 (Horrocks & Patel-Schneider, 1998). This data set demonstrates how existential quantification can lead to inefficiencies in tableau-based reasoners. Table 5.2 contains some statistics for all three benchmark sets, including the number of chains of disjunctions in super-concepts (SUPDCHN), (2) the number of disjunctions in the ontologies (DISJ) and (3) the number of existential quantifications (EF) (Kang et al., 2012, 2014). Since our encoding need the number of individuals in order to be complete, the number of individuals is calculated using the rules in Section 5.1.2.

Table 5.2: Simple ontology details for evaluation of $\mathcal{ALC}$.

Ontology	**# axioms**	**# concepts**	**# roles**	**SUPDCHN**	**DISJ**	**EF**
k_d4_sat_09.owl	722	721	1	0	0	453
k_d4_sat_10.owl	842	841	1	0	0	543
k_d4_unsat_09.owl	533	532	1	0	0	363
k_d4_unsat_10.owl	627	626	1	0	0	438
k_d4_unsat_12.owl	836	835	1	0	0	609
k_d4_unsat_13.owl	951	950	1	0	0	705
k_poly_sat_09.owl	764	763	1	0	0	520
k_poly_sat_10.owl	848	847	1	0	0	587
k_poly_sat_12.owl	1124	1123	1	0	0	812
k_poly_sat_13.owl	1328	1327	1	0	0	982
k_poly_unsat_09.owl	724	723	1	0	0	488
k_poly_unsat_10.owl	892	891	1	0	0	622
k_poly_unsat_15.owl	1606	1605	1	0	0	1217
k_poly_unsat_16.owl	1846	1845	1	0	0	1423
sat-jnh204.owl	800	100	0	77	741	0
sat-jnh212.owl	800	100	0	71	746	0
sat-jnh220.owl	800	100	0	76	747	0
unsat-jnh16.owl	850	100	0	77	795	0

[7]`http://www.MiniZinc.org`

[8]`https://zenodo.org/record/10791#.V8ZhLJN95E4`

5.3.2 Results of the Evaluation

This section presents the results of this evaluation. We proceed with the comparison of our approach w.r.t. our encoder with three selected solvers above against the dedicated ontology reasoners listed above.

Results for ORE2014 benchmark

The overview results on the ORE2014 benchmark are summarised graphically in Figure 5.1. The boxplot has been used to present these results since it can easily show the summary of the results and compare the performance of our approach and the other reasoners. The boxplot is a standardised way to present the distribution of data. It summaries five numbers: maximum, third quartile, median, first quartile, and minimum of the results. Each boxplot in Figure 5.1 uses the top line to represent maximum, the bottom line to represent minimum, the box to represent the 50% of the results (Inter-quartile range), the top line of box to represent third quartile, the middle line of box to represent median, and the bottom line of box to represent first quartile. The circles above the maximum line is called *outlier*, which is the data that seem to differ by a substantial amount from the other data. Normally, the outlier can states uncommon phenomena of the results, which we may investigate further to understand why they differ. The outlier is generally calculated as follows (Dawson, 2011):

1. Firstly, we need to determine the first quartile (Q_1) and the third (Q_3).
2. Then we calculate the interquartile range $IQR = Q_3 - Q_1$.
3. Next, we calculate the lower (LIF) and upper inner fences (UIF), $LIF = Q_1 - 1.5 \times IQR$ and $UIF = Q_3 + 1.5 \times IQR$.
4. The data points, which are below LIF or above UIF, are called *outliers*.

Note that the boxplots in Figures 5.1, 5.2, 5.3, 5.4, and 5.5 provided are based on log scale in second (s.) since the range of data is large.

As can be seen, Figure 5.1 shows that the distributions of the data for each reasoner and solver are not large. In addition, the box of each reasoner and solver shows that the maximum of reasoning time taken of each reasoner and solver is less than 1 second. This means the ontologies in this data set are easily reasoned by all reasoners and solvers. In other words, our approach is efficient for reasoning about this data set. There are some outliers of each reasoner and solver. The outliers of Konclude are not interesting since they are less than 3 seconds. However, the outliers of gecode, chuffed, and ocpx are interesting since the range of reasoning time taken by these three solvers are small, but some outliers are very high (more than 10 seconds). We investigated this further. We found that the solving time is very fast at less than 1 seconds. However, most of time taken is for flattening from MiniZinc to FlatZinc. Similarly, the outliers of HermiT are also interesting since most of reasoning time taken by HermiT is less than 1 second and the range of the results is small. We investigated those ontologies that make HermiT difficult to reason about. We found that those ontologies have high number of existential quantifications, which are one of the sources of inefficiency of tableau-based reasoners.

Moreover, Figure 5.1 shows that the performance of all solvers is similar. Interestingly, we can compare the performance of our approach against Konclude and HermiT. As can be seen, the medians of Konclude and our approach (gecode, chuffed, and ocpx) are very low at less than 0.1 second. The spread of the results of our approach and Konclude is similar. The median of HermiT is the highest one, which means that for most benchmarks, the reasoning time taken of HermiT is longer than the others.

In conclusion, our approach presented is competitive with the dedicated ontology reasoner, Konclude. Moreover, it clearly outperforms the tableau-based reasoner, HermiT. We cannot get the results from Light and the SAT approach for ORE2014 dataset since Light and the SAT approach cannot read the ontologies in this dataset.

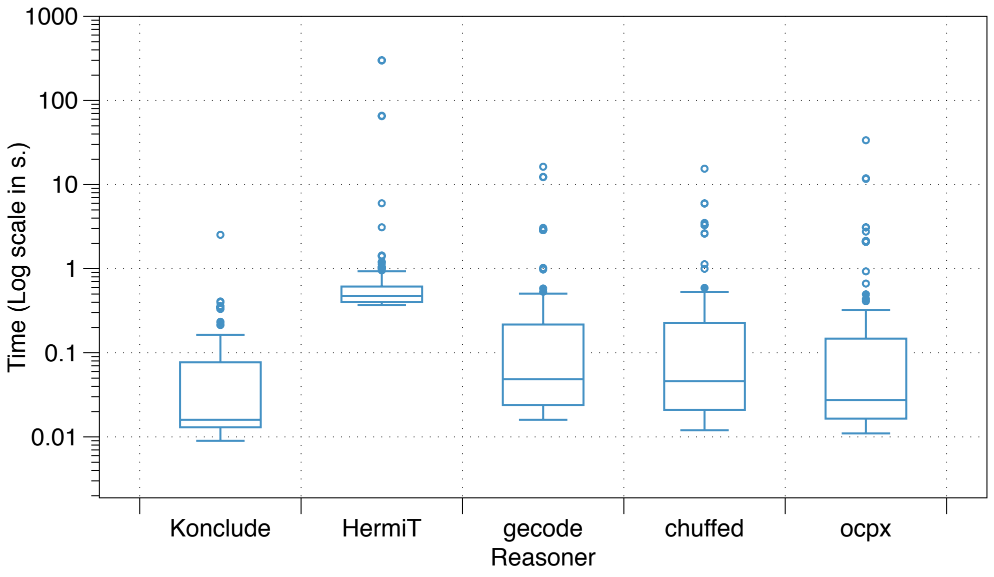


Figure 5.1: Overview of ontology consistency checking time on ORE2014.

Results for JNH benchmark

The next data set is JNH. The overview results on the JNH benchmark are summarised graphically in Figure 5.2. As described, this benchmark contains ontologies containing many disjunctions, which is one of the sources of inefficiency of the tableau-based reasoner. From Figure 5.2, it can be seen that the distributions of the results of Konclude, gecode, chuffed, and ocpx are very narrow. This evaluation is very straightforward. Our approach is much faster than Konclude and HermiT especially for the unsatisfiable ontologies (as shown in Figure 5.3). Konclude is times out for most of unsatisfiable ontologies and HermiT is almost 100 times slower than our approach on those ontologies. However, Konclude can finish reasoning on some satisfiable ontologies in this data set, but the reasoning time taken is very long and slower than our approach as shown in Figure 5.4. We further investigated the structure of the difficult ontologies in this data set. We found that all of the difficult ontologies contain numerous disjunctions and chains of disjunctions. It follows what we expected in the first place since the disjunction is difficult for the tableau-based reasoners to handle. Interestingly, it seems that the learning solvers (such as chuffed and ocpx) demonstrate their advantages on this data set.

Our approach is still a bit slower than the Light reasoner. We investigated this further. We found that the reason is our approach has overheads, which are encoding (from reading OWL ontology and encoding it to MiniZinc) and flattening (converting MiniZinc to FlatZinc) an ontology, which is around 66% of the reasoning time as shown in Table 5.4 because our parsers are standard parsers (i.e., OWLAPI and `mzn2fzn`). `mzn2fzn` is the translator for translating MiniZinc to FlatZinc. On the other hand, we believe that the main advantage of Light is pre-processing time, which is faster than our approach since Light has a more efficient parser. As mentioned, our encoding is linear time so the encoding time is linear regarding the size of ontologies. It can be seen from Table 5.4 that the encoding time and flattening time are almost constant since the size of ontologies are the same. If we only consider the solving time, the performance of our approach is competitive with that of Light. In addition, for some ontologies, our approach is even faster than Light. We believe that if the parser is optimised, the performance of our approach will be similar to that of Light or even better. For example, we may implement an efficient parser, which can read an OWL ontology efficiently and encode it to FlatZinc directly. Therefore, we may get rid of these overheads.

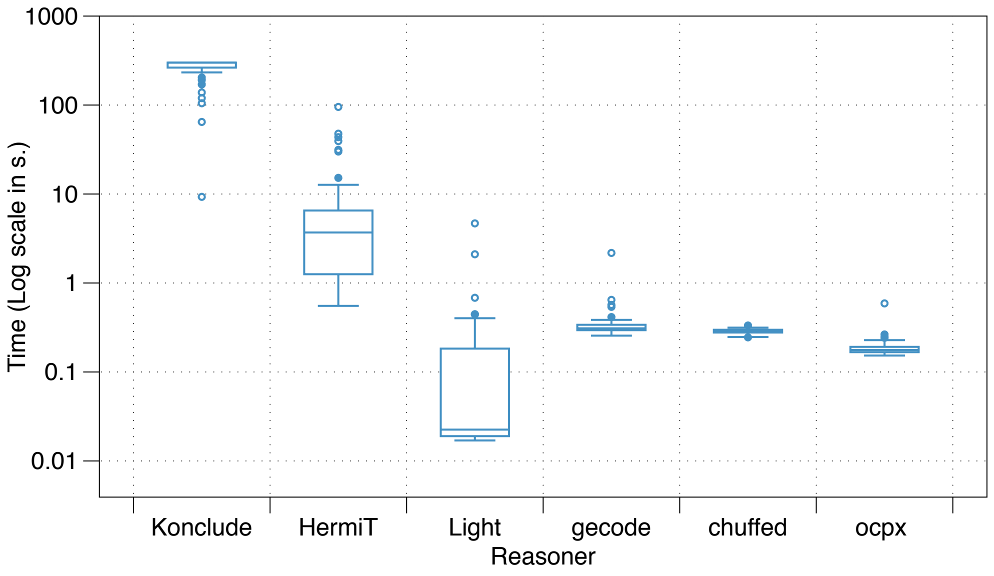


Figure 5.2: Overview of ontology consistency checking time on JNH.

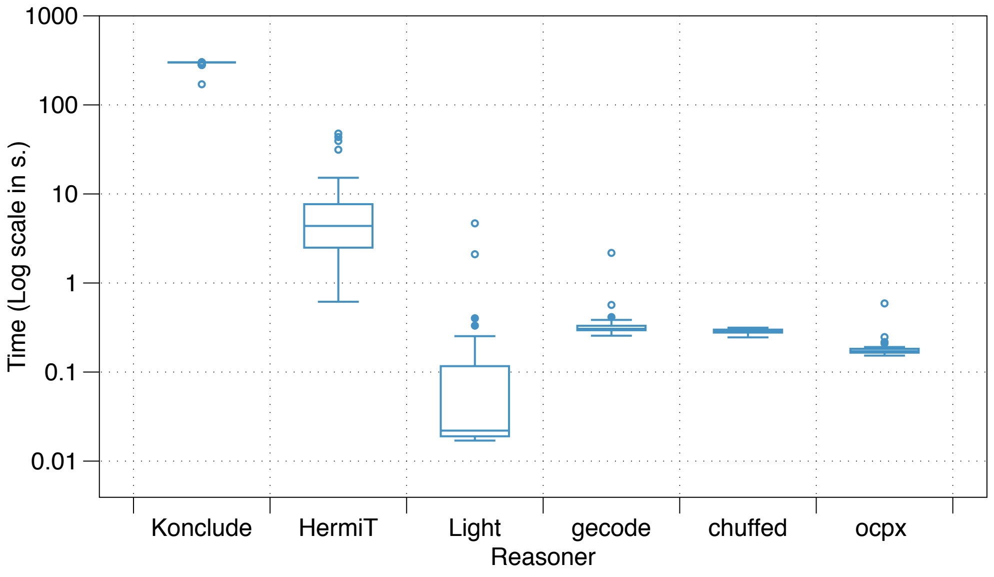


Figure 5.3: Overview of ontology consistency checking time on JNH: UNSAT ontologies.

Results for Tableaux'98 benchmark

The evaluation of the Tableaux'98 benchmark is a very good lesson for us. Only a few ontologies can be solved by our approach since the MiniZinc models of the ontologies in this benchmark require extremely high number of individuals due to the ultimately high number of chains of existential restrictions in the ontologies and the number of chains of

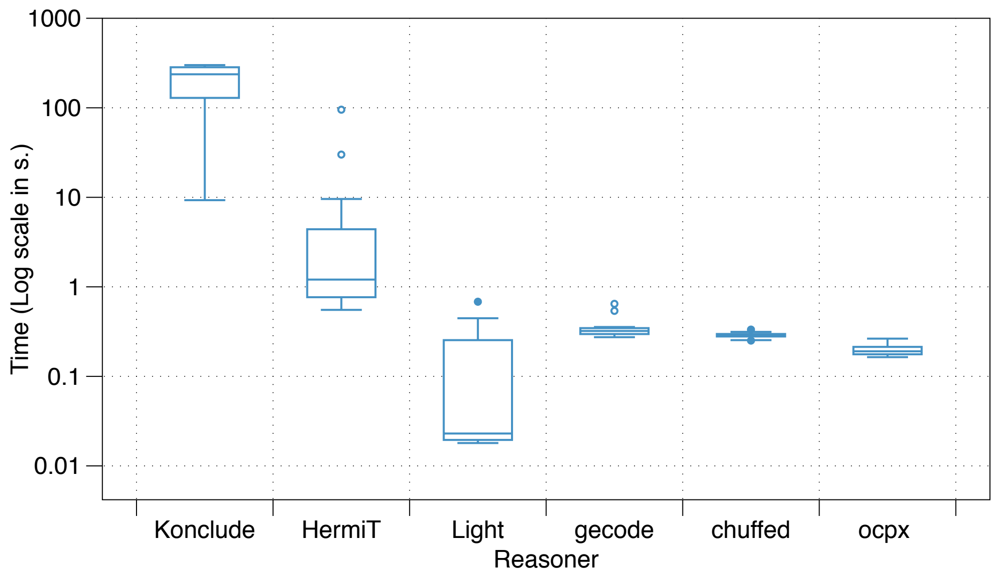


Figure 5.4: Overview of ontology consistency checking time on JNH: SAT ontologies.

Table 5.3: Details of hardest ontologies for tableau-based reasoners in JNH.

Ontology	SUPDCHN	DISJ	EF	CONJ	SAT?
sat-jnh204.owl	77	741	0	0	Y
unsat-jnh219.owl	78	724	0	0	N
unsat-jnh11.owl	81	782	0	0	N
unsat-jnh19.owl	77	788	0	0	N
sat-jnh207.owl	67	742	0	0	Y
unsat-jnh9.owl	74	790	0	0	N
unsat-jnh308.owl	78	833	0	0	N
unsat-jnh16.owl	82	795	0	0	N
unsat-jnh306.owl	79	837	0	0	N
sat-jnh212.owl	67	746	0	0	Y

existential restrictions. Table 5.5 shows samples of the number of individuals generated for the MiniZinc models of the the ontologies in this benchmark. We found that if the number of individuals of MiniZinc models is greater than 10,000, those MiniZinc models cannot be flattened from MiniZinc to FlatZinc by MiniZinc translator (`mzn2fzn`). Even though the number of individuals of MiniZinc models that is less then 10,000 is accepted by MiniZinc translator, it takes extremely long time to translate MiniZinc to FlatZinc. We found that the number of individuals of which MiniZinc models can be translated to FlatZinc and the translating time is acceptable, is less than 200.

In sprite of the problem with the number of individuals, we restrict the number of individuals for all satisfiable ontologies to 100. Our approach is sound for this benchmark in the sense that if our approach returns satisfiable, an ontology is satisfiable.

The results on the satisfiable ontologies in the Tableaux'98 benchmark are summarised graphically in Figure 5.5. As described, this benchmark contains ontologies containing many existential restrictions, which is one of the sources of inefficiency of the tableau-based reasoner. As can be seen, Figure 5.5 shows that the distributions of Konclude, HermiT, and Light are large. On the other hand, the distributions of our approach (gecode, chuffed, and ocpx) are small. Since the maximum, minimum, and median of gecode, chuffed, and ocpx are similar, we will explain them together as *our approach*. We found that the

Table 5.4: Performance comparison between Chuffed and Light.

	chuffed				Light
	Encoding	**Flattening**	**Solving**	**Total Time**	
sat-jnh1.owl	0.17	0.111	0.033	0.314	0.020
sat-jnh201.owl	0.11	0.111	0.089	0.310	0.018
sat-jnh220.owl	0.17	0.084	0.033	0.287	0.022
sat-jnh7.owl	0.17	0.105	0.023	0.298	0.019
unsat-jnh16.owl	0.15	0.106	0.056	0.312	0.253
unsat-jnh305.owl	0.11	0.110	0.081	0.301	0.402
unsat-jnh306.owl	0.16	0.111	0.043	0.314	0.029
unsat-jnh308.owl	0.15	0.113	0.040	0.303	0.020
unsat-jnh309.owl	0.15	0.109	0.047	0.306	0.027
unsat-jnh310.owl	0.12	0.086	0.096	0.302	0.023

Table 5.5: Sample number of individuals for evaluation of $\mathcal{ALC}$.

Ontology	**EF**	**# individuals**
k_d4_sat_09.owl	453	18782
k_d4_sat_10.owl	543	25614
k_d4_unsat_09.owl	363	9304
k_d4_unsat_10.owl	438	13041
k_d4_unsat_12.owl	609	23709
k_d4_unsat_13.owl	705	30980
k_poly_sat_09.owl	520	6901
k_poly_sat_10.owl	587	8183
k_poly_sat_12.owl	812	12949
k_poly_sat_13.owl	982	16973
k_poly_unsat_09.owl	488	6313
k_poly_unsat_10.owl	622	8879
k_poly_unsat_15.owl	1217	23068
k_poly_unsat_16.owl	1423	28874

combination of conjunctions and existential restrictions is difficult for the tableau-based reasoners (Konclude and HermiT) to perform reasoning for most cases. There are some outliers of each solver. We investigated this further. Then we found that the outliers of our approach are interesting since the range of reasoning time taken by these three solvers are small, but some outliers are very high (some of them are 300 seconds). It seems that this is because the combination of conjunctions and existential restrictions also has an affect on our approach for some cases.

In addition, the performance of our approach is similar to that of Light as displayed in Figure 5.5. Obviously, HermiT has very poor reasoning performance for this data set since the median is equal to the maximum data. As can be seen from Figure 5.5, the median of our approach is lower than that of Konclude. This means that the reasoning time taken by our approach on most of ontologies is faster than that of Konclude.

Now, we have presented the overall results of Tableau'98 evaluation on the satisfiable ontologies. Next, we are going to present the results in more details. We present the results in Figure 5.6 using a scatter plot. This scatter plot is ordered by the reasoning time taken by Konclude. From Figure 5.6, it can be seen that the reasoning time taken by our approach (gecode (green circle), chuffed (red circle), and ocpx (blue circle)) is pretty stable. In addition, our approach can easily reason about the ontologies that are

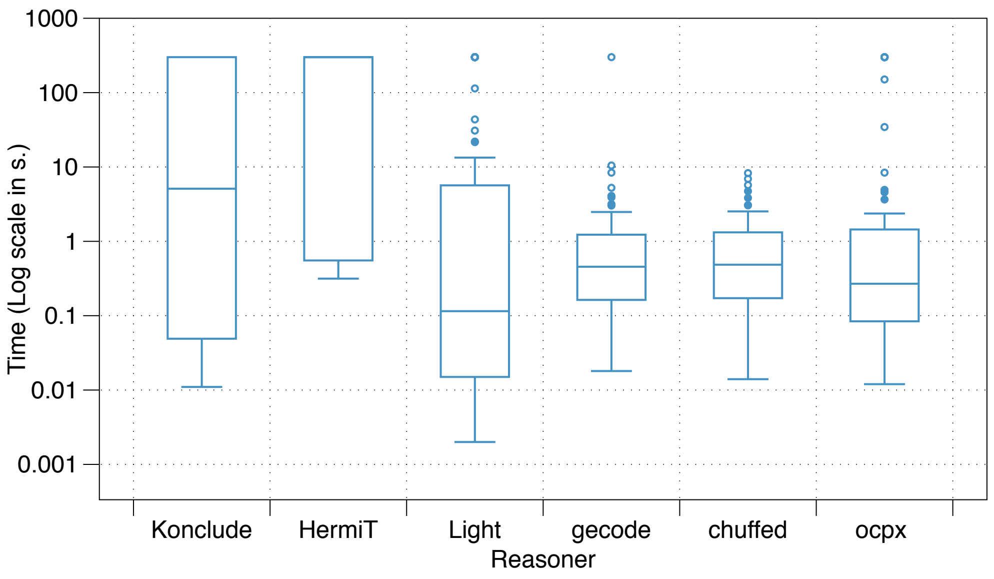


Figure 5.5: Overview of ontology consistency checking time on Tableaux'98: SAT ontologies.

hard for Konclude (orange circle) and HermiT (pink circle). Then we investigated the structure of the hard ontologies for Konclude and HermiT. We found that those hard ontologies contain a lot of existential restrictions (EF) and conjunctions (CONJ) as shown in Table 5.6. Table 5.6 presents the details of the top 6 most difficult ontologies. It follows what we expected in the first place since the combination of existential restrictions and conjunctions is hard for the tableau-based reasoners to deal with.

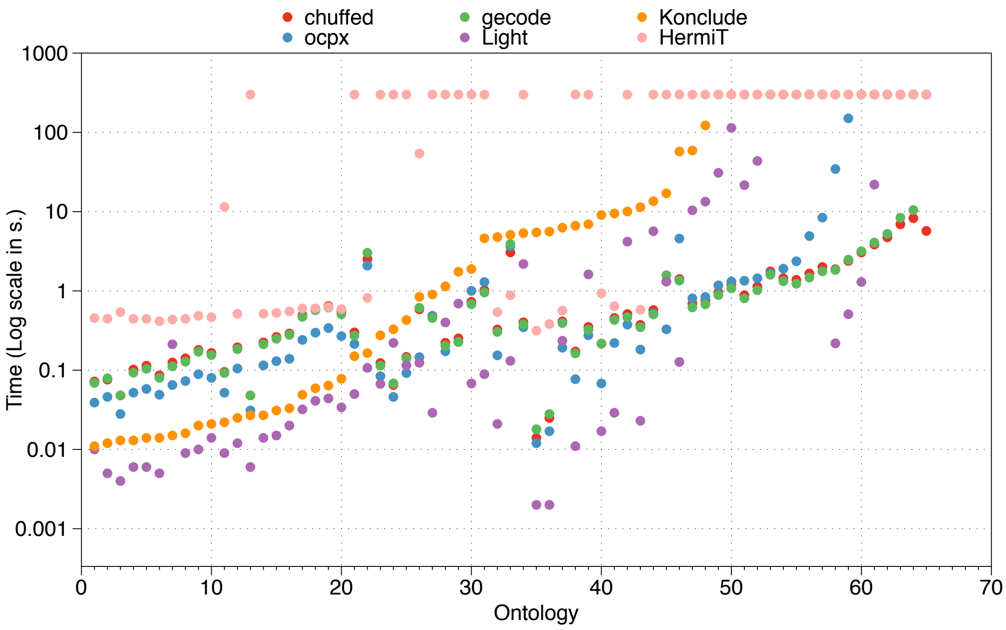


Figure 5.6: Scatter plot of ontology consistency checking time on Tableaux'98: SAT ontologies.

In conclusion, the results of this evaluation show that the approach presented in this thesis (gecode, chuffed, and ocpx) clearly outperforms the dedicated ontology reasoners. In addition, our approach is competitive with Light reasoner. This shows that for ontologies

Table 5.6: Details of hardest ontologies for tableau-based reasoners in Tableau'98: SAT ontologies.

Ontology	SUPDCHN	DISJ	EF	CONJ	SAT?
k_poly_sat_16.owl	0	0	1370	337	Y
k_poly_sat_17.owl	0	0	1588	365	Y
k_poly_sat_18.owl	0	0	1703	379	Y
k_poly_sat_19.owl	0	0	1945	407	Y
k_poly_sat_20.owl	0	0	2072	421	Y
k_poly_sat_21.owl	0	0	2338	449	Y

with a significant number of existential restrictions, tableau-based reasoners struggle to perform well. On the other hand, CP-based techniques, especially those based on clause learning (such as chuffed and ocpx) can demonstrate their advantages. Unfortunately, our approach is not completed for this benchmark. As can be seen, with restriction of number of individuals, our approach can efficiently perform reasoning on the satisfiable ontologies in this benchmark. This shows that the approach that we used to calculate the number of individuals is not efficient and MiniZinc models may not require very high number of individuals. It is interesting to further investigate the approach to tighten the number of individuals that MiniZinc models need.

While this set of benchmarks is by no means exhaustive, it demonstrates that our approach is feasible and effective, and that clause learning solvers are crucial for achieving high reasoning performance on difficult ontologies.

5.4 Related Works

Over the years, Description Logics have played an important role in many applications in numerous areas of computer science, semantic web, and ontologies. Due to this reason, the automated reasoning problem has been investigated (Schmidt-Schauß & Smolka, 1991; Baader, 2000; Horrocks & Sattler, 2007; Motik et al., 2009). Many approaches have been proposed for efficiently reasoning about Description Logics, starting from the concept satisfiability checking problem of the core Description Logic $\mathcal{ALC}$. The main algorithm is the *Tableau-based Algorithm*. This algorithm is based on tableau calculi (Baader & Nutt, 2003).

The tableau-based algorithms attempt to construct a model of an ontology as a tree structure where the branches are close by logical contradictions. These algorithms transform an input ontology to *Negation Normal Form* (NNF). For example, $\neg\exists R.(A \sqcap \neg B)$ can be transformed to $\forall R.(\neg A \sqcup B)$, where A and B are atomic concept and R is a role. After that, the algorithms apply expansion rules to construct a model of the ontology. The expansion rules are used to expand the search tree to find a contradiction (see (Baader & Sattler, 2001) for details). If there is no contradiction, the ontology is consistent. They can be deterministic and non-deterministic. Since disjunction and general concept inclusion (GCI) are handled by non-deterministic rules, they can cause a lot of OR-branchings during the reasoning process and thus decrease the performance of this type of algorithms. The application of non-deterministic rules can be considered as OR-search which is one of the major sources of inefficiency of tableau-based algorithms (Motik et al., 2007b, 2007a, 2009). The application of expansion rules on qualified number restriction and concrete domains can also increase the search space exponentially (Baader, Horrocks, & Sattler, 2008). Our approach is an alternative approach, which uses the idea to leverage the power of Constraint Programming (CP). Since CP is good at dealing with OR-search, our approach uses these advantages to minimise the inefficiency of the tableau-based reasoners. The evaluation in Section 5.3 is an evidence to support this idea since our approach outperforms the dedicated tableau-based reasoner.

Another interesting decision procedure for satisfiability checking is *Resolution-based method* (Bachmair & Ganzinger, 1997). Basically, $\mathcal{ALC}$ is a syntactic variant of modal logic (Schild, 1991). Therefore, resolution-based methods for modal logics can be used to check satisfiability of $\mathcal{ALC}$. There are many studies about resolution-based methods for modal logics (Hustadt, de Nivelle, & Schmidt, 2000; Schmidt, 2006; Schmidt & Hustadt, 2013). The widely used approach to develop resolution-based method for modal logic is to translate modal logic into first-order logic and then first-order resolution provers can be used (Hustadt et al., 2000; Schmidt, 2006; Schmidt & Hustadt, 2013). The translation from modal logic into first-order logic is straightforward, and can be obtained in time $O(n\ log\ n)$. In addition, first-order resolution can be refined to support modal logic effectively (Hustadt et al., 2000; Schmidt, 2006; Schmidt & Hustadt, 2013). Most refinements are based on ordered resolution and selection-based resolution. Since the resolution-based method is efficient for certain logics, the tableau based approach is currently the most widely used technique for reasoning in DLs (Baader, Horrocks, & Sattler, 2008).

Since the reasoning techniques offered by SAT are powerful and effective, several approaches have been proposed to exploit the advantages of SAT-style solving for DL reasoning. The first approach is *Extended Backjumping* (Steigmiller, Liebig, & Glimm, 2012). It uses a SAT-like technique to improve *Dependency Directed Backtracking* (Horrocks, 2003), which is a well-known optimisation for ontology reasoning. Moreover, Extended Backjumping exploits an *unsatisfiability caching* that caches sets of concepts, which are unsatisfiable, and *Backtracking* to reduce more of the non-deterministic branching than the original optimisation. This approach uses Backtracking to collect concepts that cause a clash and their dependencies on the same node, where the clash has occurred to generate unsatisfiability caches. Then the approach learns from the unsatisfiability caches to prune non-deterministic branching. As a result, from their experiments, this approach can reduce the number of the non-deterministic branching by two-fold compared to the original Dependency Direct Backtracking.

The second approach is an encoding approach that exploits the advantages of SAT solvers. In order to use SAT solvers, a SAT encoding approach for an ontology based on the Description Logic $\mathcal{ALCN}$ has been proposed (Meissner, 2011). This approach transforms an ontology in *negation normal form* (NNF) to propositional logic formulas in *Conjunctive Normal Form* (CNF). CNF is a conjunction of clauses. After that, they use a SAT solver to check satisfiability of the propositional logic formulas. The drawback of this approach is that it can lead to an exponential number of clauses in CNF due to the presence of number restrictions. Another encoding approach has been proposed for the Description Logic $\mathcal{ALCQ}$ (Haarslev et al., 2011). The main point of this approach is to encode an ontology based on the Description Logic $\mathcal{ALCQ}$ into a Satisfiability Modulo the Theory of Costs (SMT($\mathcal{C}$)) formula (Cimatti, Franzén, Griggio, Sebastiani, & Stenico, 2010). The Theory of Cost is used to handle *qualified number restriction.* After the transformation of the ontology is completed, the formula is sent to the SMT($\mathcal{C}$) solver to check satisfiability. If the solution is satisfiable, that means the ontology is consistenct. The issue of this approach is the large number of individuals generated regarding the at-least restrictions. However, this work also provides a technique, which is called *Smart Partitioning* to tackle the issue. As can be seen, this approach takes the advantages of the SAT and SMT solvers directly by transforming the Description Logics to propositional logic formulas, which are accepted by the solvers. For some large ontologies that contain nested qualified number restrictions and nested subsumption relationships, this approach however takes a long time to encode such ontologies to propositional logic formulas. In addition, it can be seen clearly that the encoding approach used to reduce DL problems to SAT problems has an exponential behaviour that may be not suitable for real-world DL applications. Our approach is inspired by this approach. The differences are: (1) we improve the encoding process, which is now linear time and space, and creates succinct and small MiniZinc constraint models and (2) we use CP, which can help us to extend the encoding approach to support reasoning for concrete domain and aggregations. Our approach can be easily extended to support reasoning for $\mathcal{ALCQ}$. However, since we focus on supporting reasoning on concrete domain and aggregations in this research, we leave the reasoning for $\mathcal{ALCQ}$ to future work for now.

The third approach is called *Intelligent Tableau Algorithm for DL Reasoning* (Zuo & Haarslev, 2013). This approach introduced a new normal form namely *Description Logic Normal Form* (DLNF). A TBox $\mathcal{T}$ is in DLNF if and only if it can be divided into three sets which are (1) $\mathcal{T}_g$ that contains axioms in form $\top \sqsubseteq C$ where C is a disjunction of unary literals. However, if "$\top \sqsubseteq$" is ignored, $\mathcal{T}_g$ can be viewed as propositional logic conjunctive normal form (PCNF) or the ground logic, (2) $\mathcal{T}_{ue}$ that contains axioms in form $A \sqsubseteq \exists R.C$ and (3) $\mathcal{T}_{ua}$ that contains axioms in form $A \sqsubseteq \forall R.C$, where A is an atomic concept, R is an atomic role and C is a concept. This approach is implemented in an ontology reasoner, namely *LIGHT*.

The main idea of this approach is conducting reasoning in tableau algorithm style but using a *Quantified Boolean Formulas* (QBF) solver to lead the expansion of the search tree regarding the answer (aka model) from the solver. If there are concepts in the model that are the same as the concepts of the left-hand side of the axioms in $\mathcal{T}_{ue}$ and $\mathcal{T}_{ua}$, the search tree is expanded and the concept of the right-hand side of the axioms in $\mathcal{T}_{ue}$ and $\mathcal{T}_{ua}$ are added to $\mathcal{T}_g$ of a newly created node. This approach also uses the solver to deal with the sources of inefficiency of the tableau-based algorithm such as disjunction ($\mathcal{T}_g$ set) effectively. Moreover, this approach is integrated with global learning including *Unsat-learning*, *Sat-learning* and *Unknown-learning*. This global learning can dramatically reduce the search space. Their evaluation also shows the significant improvement over previous approaches. Nonetheless, this approach supports only $\mathcal{ALC}$ and it solves only the top concept satisfiability problem. Another issue in this approach is the way that this approach feeds propositional logic formulas to the QBF solver. LIGHT needs to feed a whole set of new propositional logic formulas to the QBF solver each time. This means the learned clauses do not survive to be used in subsequent invocations of the QBF solver. Our approach translates the entire ontology into a single constraint model, so that learning solvers can take advantage of a more global perspective.

Chapter 6

A Novel Description Logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$

Description logics (DLs) (Baader & Nutt, 2003) are a family of logic-based knowledge representation formalisms that provide underlying semantics for modern ontology languages such as OWL 2 (Grau et al., 2008). Many extensions/variations have been proposed for the prototypical DL $\mathcal{ALC}$ (Schmidt-Schauß & Smolka, 1991), mostly in the abstract logical level. In many applications, *concrete domains* (Lutz et al., 2003) such as natural numbers and predicates would be useful to define concepts. For example, if we want to define students with grade *high distinction*, it might require an additional property to express that such students should have a score of at least 80. Hence, we need to introduce a new *feature* score, and define $\mathsf{HDStudent}$ by a concept desctipiton $\mathsf{Student} \sqcap \geq .(\mathsf{score}, 80)$, where $\geq .(\mathsf{score}, 80)$ means that any instance that has score greater than or equal to 80. Existing uses of concrete domain such as $\mathcal{SHOIN}^{+}(\mathbf{D})$ (Horrocks & Patel-Schneider, 2003) and $\mathcal{SHOQ}(\mathcal{G})$ (Pan, 2007) are limited in their expressivity in the concrete domain. $\mathcal{SHOIN}^{+}(\mathrm{D})$ uses the concrete domain as *built-in* data types such as $\mathsf{xsd{:}integer}$. $\mathcal{SHOQ}(\mathcal{G})$ uses the concrete domain as user-defined data type and data type predicates in ontology applications. For example, the due date of assignment on 13/03/2017 can be defined as *earlier than 13/03/2017*, where *earlier than 13/03/2017* is a user-predefined data type predicate.

Aggregation on concrete domains, such as the sum or minimum of all feature values of an individual's successors, allows more complex phenomena to be expressed. However, extensions on *concrete domains* (Lutz et al., 2003) with *aggregations* (Baader & Sattler, 2003) have not received much attention. Adding aggregation to a DL has been proposed in the logic $\mathcal{EL}(\Sigma)$ (Baader & Sattler, 2003). However, no reasoning support has been implemented for this logic. We conjecture that this is partly due to the lack of complexity results, and partly (and more importantly) due to the difficulty of extending existing, highly-optimised tableau-based algorithms.

The ability to express aggregation over concrete domains is useful in many domains and situations. For example, recently, an ontology has been used in the STAR-CITY system (Lécué et al., 2014), which supports semantic traffic analytics and reasoning. A main contribution of STAR-CITY is the ability to explain causes of traffic congestions. Adding support for concrete domain and aggregation reasoning could make an ontology model more precise and direct. For example, in a traffic ontology like STAR-CITY, concrete domains with aggregation could allow us to directly model the relationship between the number of cars on a road and the level of congestion of the road, and to reason about this relationship. Without them such relationships can only be modelled *externally* to the ontology and in an abstract way.

In this work, we develop a novel Description Logic to support concrete domain and aggregations in TBoxes. We then explore the use of modern constraint solving technologies for reasoning with this new Description Logic.

The rest of this chapter is structured as follows. In Section 6.1, we formally define the syntax and semantics of our novel Description Logic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. Then we present extensions of the OWL functional syntax, which support $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. In Section 6.3, we present the terminological reasoning problems in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ together with some examples to illustrate each reasoning problem. Next, we prove that $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is decidable. Finally, we close this chapter by discussing some related works.

6.1 $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ Syntax and Semantics

We have developed a novel Description Logic with concrete domain and aggregations named $\mathcal{ELU}^{(\neg)}(f,\Sigma)$, with some syntactic restrictions on $\mathcal{EL}(\Sigma)$ (Baader & Sattler, 2003). The main differences between $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ and $\mathcal{EL}(\Sigma)$ are listed as follows:

1. $\mathcal{EL}(\Sigma)$ does not allow TBoxes (Baader & Sattler, 2003). However, $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ allows acyclic TBoxes since the TBox is useful in many applications. This also enable more interesting reasoning services such as ontology consistency checking and and subsumption checking,
2. $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ extends $\mathcal{EL}(\Sigma)$ by including the aggregation sum,
3. The considered concrete domain of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is restricted to natural numbers,
4. Feature chains are disallowed in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$

The Description Logic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is a variation of $\mathcal{EL}(\Sigma)$ described in Section 2.5.2. $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ provides aggregation functions min, max, sum and count. In addition, concrete domains are restricted to so-called *admissible* concrete domains (Baader & Hanschke, 1991), which is defined in Section 2.5.1.

Let A and B represent concept names, $\hat{C}, \hat{D}, \ldots$ represent (anonymous) concept descriptions, R represents a role, and F_1, F_2 be features. The notation "$[\![\ ]\!]$" is used to represent a multiset. Concept descriptions in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ are defined through the concept constructs in Table 6.1, where f represents a (functional) atomic feature and $\bowtie$ represents a relational operator (a binary predicate) over the concrete domain. Features in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ are defined through the feature constructs in the middle of the Table 6.1. The semantics of aggregation functions is defined using multisets (Baader & Sattler, 2003). An $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ terminology (TBox) $\mathcal{T}$ is a finite set of axioms as defined in Table 6.1. For our work, we consider the concrete domain to be a domain of *natural number*. Therefore, we denote the concrete domain with aggregations as $\mathbb{N}$. It is interesting to investigate integers in the future.

The semantics of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is defined in the usual way, in terms of *interpretations* $\mathcal{I} = (\Delta^{\mathcal{I}}, \Delta^{\mathbb{N}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ (resp. $\Delta^{\mathbb{N}}$) is the abstract (resp. concrete) domain of $\mathcal{I}$, and $\cdot^{\mathcal{I}}$ is an interpretation function, which maps each concept name (atomic concept) A to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each role name R to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. In addition to the usual mappings, $\cdot^{\mathcal{I}}$ maps every atomic feature f to a partial function $f^{I} : \Delta^{\mathcal{I}} \to \Delta^{\mathbb{N}}$. To define the semantics of aggregation, the mapping $M_x^{(R\circ f)^{I}} : \Delta^{\mathbb{N}} \to \Delta^{\mathbb{N}}$ is used. $\Gamma(R \circ f)$ and aggregation variable $\Gamma(X)$, where X is a multiset, are defined in the same way as one in Definition 2.5.5 in Section 2.5.2. Multiset can contain an individual more than once. For example, the concept description $= .(100, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{steps}))$ means that any individual in this concept needs to have exactly 100 equal to the sum of steps-values of $\mathsf{exercise}$-successors. The multiset $\mathsf{exercise} \circ \mathsf{steps}$ is $[\![40, 40, 20]\!]$, where there are three

Table 6.1: The syntax and semantics of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$.

Concepts	**Syntax**	**Semantics**
top concept	$\top$	$\Delta^{\mathcal{I}}$
concept negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
atomic concept	A	$A^{\mathcal{I}}$
conjunction	$\hat{C} \sqcap \hat{D}$	$\hat{C}^{\mathcal{I}} \cap \hat{D}^{\mathcal{I}}$
disjunction	$\hat{C} \sqcup \hat{D}$	$\hat{C}^{\mathcal{I}} \cup \hat{D}^{\mathcal{I}}$
existential restriction	$\exists R.\hat{C}$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in \hat{C}^{\mathcal{I}}\}$
concrete domain	$\bowtie .(F_1, F_2)$	$\{x \in \Delta^{\mathcal{I}} \mid \exists d_1, d_2 \in \Delta^{\mathbb{N}} : (x,d_1) \in F_1^I \wedge (x,d_2) \in F_2^I \wedge (d_1,d_2) \in \bowtie^{\mathbb{N}}\}$, where $\bowtie \in \{\geq, <, \leq, >, =, \neq\}$ and $\bowtie^{\mathbb{N}} \subseteq \Delta^{\mathbb{N}} \times \Delta^{\mathbb{N}}$
Features	**Syntax**	**Semantics**
atomic feature	f	$\Delta^{\mathcal{I}} \to \Delta^{\mathbb{N}}$
natural number	d	$d^{\mathbb{N}}$
aggregation	$\Gamma(R \circ f)$	$\begin{cases} \Gamma^{\mathbb{N}}(M_x^{(R\circ f)^I}), & \text{if } M_x^{(R\circ f)^I} \text{is a multiset} \\ \text{undefined}, & \text{otherwise} \end{cases}$ where $M_x^{(R\circ f)^I} = [\![d \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge f^{\mathcal{I}}(y) = d]\!]$ and $\Gamma \in \{\mathsf{sum}, \mathsf{count}, \mathsf{min}, \mathsf{max}\}$
Axioms	**Syntax**	**Semantics**
concept inclusion	$A \sqsubseteq \hat{D}$	$A^{\mathcal{I}} \subseteq \hat{D}^{\mathcal{I}}$
concept definition	$A \equiv \hat{D}$	$A^{\mathcal{I}} = \hat{D}^{\mathcal{I}}$

exercise-successors and each successor has steps-value 40, 40, and 20 respectively. Thus, the sum value of the multiset $\mathsf{exercise} \circ \mathsf{steps}$ ($\mathsf{sum}(\mathsf{exercise} \circ \mathsf{steps})$) is 100.

In order to retain decidability of ontology consistency checking, some syntactic restrictions are placed on $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. The syntactic restrictions are:

1. The TBox must be acyclic (see Definition 6.1.1).

2. Negation is only allowed in front of atomic concepts on the right-hand side of subsumption axioms. Moreover, existentially constrained concepts cannot be negated. For example, $\neg A$ is not allowed if $A \sqsubseteq \exists R.B$ is in the TBox.

3. Each feature needs to have a finite and known range of values. E.g., for a feature weight, $0 \leq \mathsf{weight}^{\mathcal{I}}(x) \leq d$ for any individual x, where $d \in \mathbb{N}$ is a constant. It can be expressed as $\top \sqsubseteq \leq .(0, \mathsf{weight}) \sqcap \leq .(\mathsf{weight}, d)$

4. For each role R involved in some aggregation, the number of R-successors need to be bounded. It can be expressed as $\top \sqsubseteq \leq .(\mathsf{count}(R \circ f), d)$, where $d \in \mathbb{N}$ is a constant and f is an arbitrary feature.

5. The top concept ($\top$) can be only used as shown in Restrictions 3 and 4.

Restriction 1 helps ensure that the number of necessary individuals for the abstract part is finite. Another reason of Restriction 1 is that $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ cannot allow negation to occur in front of atomic concepts on the left-hand side of axioms, because this could be used to force universal restrictions on feature values. For example, the TBox $A \sqsubseteq \geq$

$.(f,5), \neg A \sqsubseteq \geq .(f,5)$ forces all f-values to be at least 5, which would not be part of the procedure checking for consistency of the concrete domain. With the axiom syntax defined in Table 6.1, where negation cannot be on the left-hand side of axioms, new $R \circ f$-successors can be safely added outside any named concepts, where they cannot affect satisfiability of any of the axioms. For example, consider the following TBox.

$$\begin{aligned} A &\sqsubseteq \geq .(f,5) \\ B &\sqsubseteq \geq .(f,5) \\ \neg A &\sqsubseteq B \\ C &\sqsubseteq \exists R.(= .(f,5)) \sqcap = .(\mathsf{sum}(R \circ f), 6) \end{aligned}$$

Now, to check TBox consistency, we need to put all concepts on the left-hand side to the right-hand side of axioms. Then combining them together as follow:

$$\top \sqsubseteq (A \sqcup B) \sqcap (\neg A \sqcup \geq .(f,5)) \sqcap (\neg B \sqcup \geq .(f,5)) \sqcap (\neg C \sqcup (\exists R.(= .(f,5)) \sqcap = .(\mathsf{sum}(R \circ f), 6))$$

Then we can check TBox consistency by checking whether $\top$ is satisfiable. According to the concept $\exists R.(= .(f,5))$, there is one R-successor with f-value of 5. In order to satisfy the concept $= .(\mathsf{sum}(R \circ f), 6)$, one additional R-successor with f-value of 1 ($R \circ f$-successor) is needed. However, the f-value of the additional R-successor can be only greater than or equal to 5 since the additional R-successor need to belong to either A or B because $\neg A \sqsubseteq B$. Since the idea of checking the TBox consistency is to conduct abstract domain reasoning and then switch to concrete domain reasoning once, the inconsistency may not be detected. $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ disallows the axiom $\neg A \sqsubseteq B$. As a result, the additional $R \circ f$-successor can be safely added. Restriction 2 prevents occurrences of universal quantification, which is not supported in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. Restrictions 3 and 4 ensure that all features and roles have finite domains, which enables us to show decidability in Lemma 6.4.4.

Definition 6.1.1. A TBox $\mathcal{T}$ is called *acyclic* if

1. there are no cyclic dependencies between its concept names, i.e., concept names are neither defined directly or indirectly in terms of themselves through axioms in $\mathcal{T}$.
2. $\mathcal{T}$ only contains axioms of the form $A \sqsubseteq \hat{D}$ and $A \equiv \hat{D}$, where A is a concept name and $\hat{D}$ is a concept description.
3. for any axiom $A \equiv \hat{D}$, $\mathcal{T}$ cannot contain any other definition for A.

Now, let us use Example 6.1.1 to shows a simple TBox in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$.

Example 6.1.1. From Figure 6.1, let HouseA, HouseB, and $\mathsf{StreetA}$ be concept names. has represents a role name and $\mathsf{numberofhouses}$ and $\mathsf{numberofcars}$ represent feature names.

This example presents knowledge about house and street in term of number of cars and number of houses. HouseA is a house that has 1 - 3 cars and HouseB is a house that has 4 - 6 cars, defined using the concrete domain concept description. Consider the axiom $\mathsf{HouseA} \equiv \geq .(\mathsf{numberofcars}, 1) \sqcap \leq .(\mathsf{numberofcars}, 3)$, $\geq .(\mathsf{numberofcars}, 1)$ means that the number of cars is at least 1.

Moreover, $\mathsf{StreetA}$ is a street that has HouseA and HouseB, the number of houses on this street is between 2 and 10, and the number of cars on this street is 9. Here, the concrete domain construct is used to define the concepts $= .(\mathsf{numberofhouses}, \mathsf{count}(\mathsf{has} \circ \mathsf{numberofcars}))$, which defines the number of houses on $\mathsf{StreetA}$, and $= .(\mathsf{numberofcars}, \mathsf{sum}(\mathsf{has} \circ \mathsf{numberofcars}))$, which defines the number of cars on $\mathsf{StreetA}$, regarding has-successors that have $\mathsf{numberofcars}$-value. Let us assume that there are two has-successors; one is for HouseA

with $\mathsf{numberofcars}$-value of 3 and another is for HouseB with $\mathsf{numberofcars}$-value of 6. Thus, the multiset $\mathsf{has} \circ \mathsf{numberofcars}$ is $[\![3, 6]\!]$. Regarding concepts $= .(\mathsf{numberofhouses}, \mathsf{count}(\mathsf{has} \circ \mathsf{numberofcars}))$ and $= .(\mathsf{numberofcars}, \mathsf{sum}(\mathsf{has} \circ \mathsf{numberofcars}))$, the number of houses is 2 and the number of cars is 9 respectively. □

$$
\begin{aligned}
\mathsf{HouseA} &\equiv \geq .(\mathsf{numberofcars}, 1) \sqcap \leq .(\mathsf{numberofcars}, 3)\\
\mathsf{HouseB} &\equiv \geq .(\mathsf{numberofcars}, 4) \sqcap \leq .(\mathsf{numberofcars}, 6)\\
\mathsf{StreetA} &\equiv \exists\ \mathsf{has.HouseA}\ \sqcap \exists\ \mathsf{has.HouseB}\\
\mathsf{StreetA} &\equiv = .(\mathsf{numberofhouses}, \mathsf{count}(\mathsf{has} \circ \mathsf{numberofcars})) \sqcap\\
&\quad \geq .(\mathsf{count}(\mathsf{has} \circ \mathsf{numberofcars}), 2) \sqcap\\
&\quad \leq .(\mathsf{count}(\mathsf{has} \circ \mathsf{numberofcars}), 10)\\
\mathsf{StreetA} &\equiv = .(\mathsf{numberofcars}, \mathsf{sum}(\mathsf{has} \circ \mathsf{numberofcars})) \sqcap\\
&\quad = .(\mathsf{sum}(\mathsf{has} \circ \mathsf{numberofcars}), 9)
\end{aligned}
$$

Figure 6.1: An example ontology about house and street

Remark 6.1.1. Since the concrete domain is finite, it is possible to emulate concrete values and their aggregations using new concept names. However, we aim to develop effective modelling and reasoning support, and such an emulation may lead to an inefficient, exponential-size encoding of the original TBox, in particular when emulating aggregation. Let us use the following axioms (part of Figure 6.1) as an example.

$$
\begin{aligned}
\mathsf{HouseA} &\equiv \geq .(\mathsf{numberofcars}, 1) \sqcap \leq .(\mathsf{numberofcars}, 3)\\
\mathsf{HouseB} &\equiv \geq .(\mathsf{numberofcars}, 4) \sqcap \leq .(\mathsf{numberofcars}, 6)\\
\mathsf{StreetA} &\equiv \exists\ \mathsf{has.HouseA}\ \sqcap \exists\ \mathsf{has.HouseB}\\
\mathsf{StreetA} &\equiv = .(\mathsf{numberofcars}, \mathsf{sum}(\mathsf{has} \circ \mathsf{numberofcars}))
\end{aligned}
$$

For the axioms $\mathsf{HouseA} \equiv \geq .(\mathsf{numberofcars}, 1) \sqcap \leq .(\mathsf{numberofcars}, 3)$ and $\mathsf{HouseB} \equiv \geq .(\mathsf{numberofcars}, 4) \sqcap \leq .(\mathsf{numberofcars}, 6)$, the concrete domain constructs in the axioms can be emulated to the following axioms:

$$
\begin{aligned}
\mathsf{HouseA} \equiv\ & \mathsf{HouseANumberofCarEqualTo1}\ \sqcup \mathsf{HouseANumberofCarEqualTo2} \sqcup\\
& \mathsf{HouseANumberofCarEqualTo3}\\
\mathsf{HouseB} \equiv\ & \mathsf{HouseBNumberofCarEqualTo4}\ \sqcup \mathsf{HouseBNumberofCarEqualTo5} \sqcup\\
& \mathsf{HouseBNumberofCarEqualTo6}
\end{aligned}
$$

where $\mathsf{HouseANumberofCarEqualToX}$ is an unique concept name for each feature value X.

Then we need to create new concept names regarding the number of combinations of number of cars of each house to handle the aggregation $\mathsf{sum}(\mathsf{has} \circ \mathsf{numberofcars})$ as follows:

$$
\begin{aligned}
\mathsf{Sum14} &\equiv \mathsf{HouseANumberofCarEqualTo1} \sqcap \mathsf{HouseBNumberofCarEqualTo4}\\
\mathsf{Sum15} &\equiv \mathsf{HouseANumberofCarEqualTo1} \sqcap \mathsf{HouseBNumberofCarEqualTo5}\\
\mathsf{Sum16} &\equiv \mathsf{HouseANumberofCarEqualTo1} \sqcap \mathsf{HouseBNumberofCarEqualTo6}\\
\mathsf{Sum24} &\equiv \mathsf{HouseANumberofCarEqualTo2} \sqcap \mathsf{HouseBNumberofCarEqualTo4}\\
\mathsf{Sum25} &\equiv \mathsf{HouseANumberofCarEqualTo2} \sqcap \mathsf{HouseBNumberofCarEqualTo5}\\
\mathsf{Sum26} &\equiv \mathsf{HouseANumberofCarEqualTo2} \sqcap \mathsf{HouseBNumberofCarEqualTo6}\\
\mathsf{Sum34} &\equiv \mathsf{HouseANumberofCarEqualTo3} \sqcap \mathsf{HouseBNumberofCarEqualTo4}\\
\mathsf{Sum35} &\equiv \mathsf{HouseANumberofCarEqualTo3} \sqcap \mathsf{HouseBNumberofCarEqualTo5}\\
\mathsf{Sum36} &\equiv \mathsf{HouseANumberofCarEqualTo3} \sqcap \mathsf{HouseBNumberofCarEqualTo6}\\
\mathsf{StreetA} &\equiv \exists\ \mathsf{has.HouseA} \sqcap \exists\ \mathsf{has.HouseB} \sqcap\\
&\quad (\mathsf{Sum14} \sqcup \mathsf{Sum15} \sqcup \mathsf{Sum16} \sqcup\\
&\quad \mathsf{Sum24} \sqcup \mathsf{Sum25} \sqcup \mathsf{Sum26} \sqcup\\
&\quad \mathsf{Sum34} \sqcup \mathsf{Sum35} \sqcup \mathsf{Sum36})
\end{aligned}
$$

where SumXY is the sum of number of cars X of HouseA and Y of HouseB.

As can be seen, the emulation makes TBox very large. Such emulations are tedious, inelegant, and expensive. □

6.1.1 An $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ Normal Form

An $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ concept description is in *negative normal form* (NNF) when negation appears only in front of concept names. A concept description $\hat{C}$ can be in NNF by applying the transformation TR1, TR4, and TR5 in Section 2.4.2 and the normalisation NR4, NR6, NR7, and NR9 in Section 2.4.1 and 2.4.2. For a concrete domain concept description $(\bowtie .(F_1, F_2))$, it is possible to obtain an equivalent NNF concept by negating predicate, i.e., $\neg(= .(F_1, F_2)) \rightsquigarrow \neq .(F_1, F_2)$. Since only acyclic TBox defined is allowed, it is sufficient to leave the concept definition as it is.

Then we restrict our attention to those $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBoxes in normal form, where all axioms are of the following forms:

$$
\begin{array}{ll}
A \sqsubseteq B & \\
A \sqsubseteq \sqcup_i B_i & A \sqsubseteq \exists R.B\\
A \sqsubseteq \bowtie .(F_1, F_2) & A \equiv \sqcap_i B_i\\
A \equiv \sqcup_i B_i & A \equiv \exists R.B\\
A \equiv \bowtie .(F_1, F_2) &
\end{array}
$$

where A, A_i, B, B_i are concept names. Note that only B can be concept negation, but not A according to the axiom syntax defined in Table 6.2.

6.2 OWL Functional Syntax Extension for $\mathcal{ELU}^{(\neg)}(f,\Sigma)$

In this section, we will present the functional syntax extension of OWL to support our logic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$, presented in Table 6.2. The syntax of top concept, concept negation, conjunction, disjunction, existential quantification, and role are the same as the standard

OWL functional syntax. The concrete domain and aggregation tie with ClassExpression in the the standard OWL functional syntax.

We extend the OWL functional syntax to support concrete domain, feature, and aggregation. The OWL functional syntax of concrete domain is 'ConcreteDomain' '(' Predicate FeatureExpression FeatureExpression ')'. The Predicate is defined as follows:

Predicate :='GreaterThanOrEqual' | 'GreaterThan' | 'LessThanOrEqual' | 'LessThan' | 'Equal' | 'NotEqual'

The meaning of predicate is vary straightforward. Each predicate is corresponding to arithmetic operators. For example, GreaterThanOrEqual means $\geq$.

The FeatureExpression can be atomic feature, natural number, or aggregation. It is defined as follows:

FeatureExpression := Feature | NaturalNumber | Aggregation

An atomic feature can be expressed using Feature. A natural number can be expressed using NaturalNumber. The syntax of Aggregation is

'Aggregation' '(' Function ObjectPropertyExpression Feature ')'

The aggregation consists of an aggregation function (Function), a role (ObjectPropertyExpression), and an atomic feature (Feature). The Function can be sum, count, max, or min.

Function := 'sum' | 'count' | 'max' | 'min'

Table 6.2: OWL functional syntax of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$.

Concepts	**Syntax**	**OWL Functional Syntax**
top concept	$\top$	owl:Thing
concept negation	$\neg A$	'ObjectComplementOf' '(' Class ')'
atomic concept	A	Class
conjunction	$C \sqcap D$	'ObjectIntersectionOf' '(' ClassExpression ClassExpression {ClassExpression} ')'
disjunction	$C \sqcup D$	'ObjectUnionOf' '(' ClassExpression ClassExpression {ClassExpression} ')'
existential quantification	$\exists R.C$	'ObjectSomeValuesFrom' '(' ObjectPropertyExpression ClassExpression ')'
concrete domain	$\bowtie .(F_1, F_2)$	'ConcreteDomain' '(' Predicate FeatureExpression FeatureExpression ')'
Role & Features	**Syntax**	**OWL Functional Syntax**
role	R	ObjectPropertyExpression
atomic feature	f	Feature
natural number	n	any natural number
aggregation	$\Gamma(R \circ f)$	'Aggregation' '(' Function ObjectPropertyExpression Feature ')'

Let us illustrate an example inspired by (Lécué, 2012) with the extended OWL functional syntax.

$$\begin{aligned}
\top &\sqsubseteq\ \geq .(\mathsf{numberofcars}, 1) \sqcap \leq .(\mathsf{numberofcars}, 1000) \sqcap \\
&\quad \leq .(\mathsf{count}(\mathsf{intersectWith} \circ \mathsf{numberofcars}), 3) && \text{(A1)}\\
\mathsf{RoadA} &\sqsubseteq\ = .(\mathsf{numberofcars}, 100) && \text{(A2)}\\
\mathsf{RoadB} &\sqsubseteq\ = .(\mathsf{numberofcars}, 200) && \text{(A3)}\\
\mathsf{RoadC} &\sqsubseteq\ = .(\mathsf{numberofcars}, 300) && \text{(A4)}\\
\mathsf{JammedRoad} &\sqsubseteq\ \geq .(\mathsf{numberofcars}, 500) && \text{(A5)}\\
\mathsf{RoadD} &\sqsubseteq \exists\ \mathsf{intersectWith}.\mathsf{RoadA} \sqcap \exists\ \mathsf{intersectWith}.\mathsf{RoadB} \sqcap \\
&\quad \exists\ \mathsf{intersectWith}.\mathsf{RoadC} \sqcap \leq .(\mathsf{numberofcars}, 600) \sqcap \\
&\quad = .(\mathsf{numberofcars}, \mathsf{sum}(\mathsf{intersectWith} \circ \mathsf{numberofcars})) && \text{(A6)}
\end{aligned}$$

Figure 6.2: An ontology about the intersection of 4 roads in DL syntax.

The ontology in Figure 6.2 represents a knowledge base about the intersection of 4 roads (RoadA, RoadB, RoadC, and RoadD) and the number of cars on a road, represented by the feature numberofcars. RoadA is a road with 100 cars on it. The number of cars on RoadB is 200. RoadC is a road with 300 cars on it. RoadD intersects with RoadA, RoadB, and RoadC. In addition, the number of cars on RoadD is the sum of the numbers of cars on RoadA, RoadB, and RoadC, and is equal to 600. JammedRoad is a road that has more than 500 cars. We can express it in the extended OWL functional syntax as shown in Figure 6.3.

Concepts and roles are declared as usual. However, regarding Restriction 3, the range of feature needs to be finite. Therefore, features need to be declared with their minimum value and maximum value. Features are declared using the following syntax:

'Declaration' '(' 'ObjectFeature' '(' Feature NaturalNumber NaturalNumber ')' ')'

where the first natural number is minimum value and the second natural number is maximum value.

```
Declaration(ObjectFeature(numberofcars 1 1000))
SubClassOf(owl:Thing ConcreteDomain(LessThanOrEqual Aggregation(sum intersectWith numberofcars) 3)
SubClassOf(RoadA ConcreteDomain(Equal numberofcars 100)
SubClassOf(RoadB ConcreteDomain(Equal numberofcars 200)
SubClassOf(RoadC ConcreteDomain(Equal numberofcars 300)
SubClassOf(JammedRoad ConcreteDomain(GreaterThanOrEqual numberofcars 500)
SubClassOf(RoadD ObjectIntersectionOf(ObjectSomeValuesFrom(intersectWith RoadA)
          ObjectSomeValuesFrom(intersectWith RoadB)
          ObjectSomeValuesFrom(intersectWith RoadC)
          ConcreteDomain(LessThanOrEqual  numberofcars 600)
          ConcreteDomain(Equal  numberofcars Aggregation(sum  intersectWith numberofcars)))
```

Figure 6.3: OWL functional syntax of the ontology about the intersection of 4 roads.

6.3 $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ Terminological Reasoning

The key TBox reasoning services for Description Logic are *concept satisfiability checking*, *ontology consistency checking*, and *concept subsumption checking*. The definition of these tasks is described in Section 2.3. This section presents *concept satisfiability*, *ontology consistency checking*, and *concept subsumption checking* that arise from $\mathcal{ELU}^{(\neg)}(f, \Sigma)$.

6.3.1 Concept Satisfiability Checking

Given an $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ TBox $\mathcal{T}$, the *concept satisfiability* task is to ensure that a particular concept is satisfiable w.r.t $\mathcal{T}$, where $\mathcal{T}$ is consistent. A concept description $\hat{C}$ is satisfiable w.r.t $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty. Note that concept satisfiability is sometimes considered with empty TBox, i.e., the concept description alone is considered. In such a case, an arbitrary interpretation that makes $\hat{C}$ non-empty is considered. Let us use Example 6.3.1 to illustrate concept satisfiability in $\mathcal{ELU}^{(\neg)}(f, \Sigma)$.

Example 6.3.1. The example ontology in Figure 6.2 helps to presents an inference problem, *concept satisfiability.* Let us now reason about the satisfiability of concept RoadD. Assume that RoadD intersects with each RoadA, RoadB, and RoadC once. Under this assumption, in order to check the satisfiability of concept RoadD, we can check the satisfiability of concept

$$\exists\ \mathsf{intersectWith.RoadA} \sqcap\ \exists\ \mathsf{intersectWith.RoadB}\ \sqcap\ \exists\ \mathsf{intersectWith.RoadC} \sqcap$$
$$\leq .(\mathsf{numberofcars}, 600) \sqcap = .(\mathsf{numberofcars}, \mathsf{sum}(\mathsf{intersectWith} \circ \mathsf{numberofcars}))$$

in A7. We need to calculate the sum of the numbers of cars on each road intersected with RoadD. Since the sum of the numbers of cars on RoadA, RoadB, and RoadC is 600 (100 + 200 + 300), the concept $\leq .(\mathsf{numberofcars}, 600)$, which states that the number of cars on RoadD is less than or equal to 600, is satisfiable. Thus, concept RoadD is satisfiable.

Alternatively, if we change axiom the concept $\leq .(\mathsf{numberofcars}, 600)$ in the right-hand side of A6 to $\leq .(\mathsf{numberofcars}, 400)$, which states that the number of cars on RoadD is less than or equal to 400, it is easy to see that the sum of the numbers of cars on RoadA, RoadB, and RoadC is greater than 400. Therefore, concept RoadD would be unsatisfiable. □

6.3.2 Limited Concept Subsumption Checking

Generally, *concept subsumption checking* is used to check whether some concept is more general than another one. Given an $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ TBox $\mathcal{T}$, a concept description $\hat{C}$ is subsumed by a concept description $\hat{D}$ w.r.t $\mathcal{T}$ if $\hat{C}^{\mathcal{I}} \subseteq \hat{D}^{\mathcal{I}}$ is true for all models $\mathcal{I}$ of $\mathcal{T}$. In this case, a concept description $\hat{C}$ is subsumed by a concept description $\hat{D}$ w.r.t $\mathcal{T}$ can be written by $\hat{C} \sqsubseteq_{\mathcal{T}} \hat{D}$ or $\mathcal{T} \models \hat{C} \sqsubseteq \hat{D}$. Concept subsumption of $\hat{C} \sqsubseteq \hat{D}$ w.r.t. $\mathcal{T}$ can be reduced to concept satisfiability of its negation $\hat{C} \sqcap \neg\hat{D}$ w.r.t. $\mathcal{T}$. $\hat{C} \sqsubseteq_{\mathcal{T}} \hat{D}$ if and only if $\hat{C} \sqcap \neg\hat{D}$ is unsatisfiable w.r.t. $\mathcal{T}$.

Since $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ is not closed under negation, we can only perform *limited* concept subsumption checking. The restrictions in Section 6.1 are not applied for the concepts that we need to check subsumption. One additional restriction is that the right-hand side of axiom ($\hat{D}$) cannot contain the existential quantification ($\exists$) because concept subsumption checking is performed by checking the satisfiability of $\hat{C} \sqcap \neg\hat{D}$.

The following example ontology about daily fitness in Figure 6.4 and 6.5 helps to illustrate the language constructs of $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ and motivates an inference problem, limited *concept subsumption checking*, that arises from $\mathcal{ELU}^{(\neg)}(f, \Sigma)$.

Example 6.3.2. In this example, we consider a situation where there is a static ontology (Figure 6.4), which defines the details about exercise machines (Treadmill, FlexStrider, and CrossTrainer) and the goal (GoalState) that a person wants to achieve. For example, Treadmill is an exercise machine that a person may use for one hour (represented by feature hours) to get 4,000 steps (represented by feature steps) and 800 calories burnt (represented by feature calburn). FlexStrider is an exercise machine that a person may use for one hour to get 3,000 steps and 700 calories burnt. CrossTrainer is an exercise machine that a person may use for one hour to get 3,000 steps and 750 calories burnt. The goal (GoalState) of a person is a state that a person spends time to exercise between 3 and 5 hours, gets more than 10,000 steps, burns more than 2,000 calories, and achieves a maximum heart rate (represented by feature HR) of has-successors of at least 128 per minute. Axiom A1 is used to restrict the range of features and limit the number of R-successors of each R that is involved in aggregations (i.e., sum) for restrictions 3 and 4 discussed above.

Suppose a dynamic ontology is used to sync a person's fitness activities in a streaming fashion, for instance using a wearable fitness tracking device. The dynamic ontology is then combined with the static base ontology and inference is performed to decide whether the daily exercise goal is achieved. In this example, we fix a window size of synced activities to 3 (i.e., three fitness activities per day).

$$\begin{aligned}
\top \sqsubseteq\ & \geq.(\mathsf{hours},1) \sqcap \leq.(\mathsf{hours},10) \sqcap \geq.(\mathsf{steps},1) \sqcap \leq.(\mathsf{steps},20) \sqcap \\
& \geq.(\mathsf{calburn},100) \sqcap \leq.(\mathsf{calburn},5000) \sqcap \geq.(\mathsf{HR},100) \sqcap \leq.(\mathsf{HR},170) \sqcap \\
& \leq.(\mathsf{count}(\mathsf{has}\circ\mathsf{hours}),3) \sqcap \leq.(\mathsf{count}(\mathsf{exercise}\circ\mathsf{hours}),5) && \text{(A1)}\\
\mathsf{Treadmill} \equiv\ & =.(\mathsf{hours},1) \sqcap \geq.(\mathsf{steps},4) \sqcap \geq.(\mathsf{calburn},800) && \text{(A2)}\\
\mathsf{FlexStrider} \equiv\ & =.(\mathsf{hours},1) \sqcap \geq.(\mathsf{steps},3) \sqcap \geq.(\mathsf{calburn},700) && \text{(A3)}\\
\mathsf{CrossTrainer} \equiv\ & =.(\mathsf{hours},1) \sqcap \geq.(\mathsf{steps},3) \sqcap \geq.(\mathsf{calburn},750) && \text{(A4)}\\
\mathsf{GoalState} \equiv\ & \geq.(\mathsf{hours},3) \sqcap \leq.(\mathsf{hours},5) \sqcap \\
& \geq.(\mathsf{steps},10) \sqcap \geq.(\mathsf{calburn},2000) \sqcap \\
& =.(\mathsf{HR},\mathsf{max}(\mathsf{has}\circ\mathsf{HR})) \sqcap \geq.(\mathsf{max}(\mathsf{has}\circ\mathsf{HR}),128) \sqcap \\
& =.(\mathsf{count}(\mathsf{has}\circ\mathsf{HR}),3) && \text{(A5)}
\end{aligned}$$

Figure 6.4: A static $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ ontology about daily fitness that describes basic knowledge of the fitness exercises.

The stream ontology (Figure 6.5) contains three activities (StateA, StateB, and StateC) and the daily aggregation (DailyAggregate), which combines the three sub activities. The id feature is used to identify the activity. StateA, for instance, is an activity that a person has exercised using Treadmill with a heart rate of 100. When an hour is spent, the steps and calories burnt are equal to the sum of those values given in Treadmill.

We can now reason about whether a person achieves her goal, i.e., whether the combined ontology entails DailyAggregate $\sqsubseteq$ GoalState. Assuming that an individual in DailyAggregate has only one of each StateA, StateB, and StateC, it can be easily seen that axioms A2–A9 are satisfiable. In order to check the subsumption, we test whether it is possible that this individual is *not* contained in GoalState. Calculating the total hours spent, the total calories burnt, the total number of steps, and the maximum heart rate of the day (DailyAggregate), we obtain 3 hours, 2,250 calories burnt, 10,000 steps, and heart rate 128, respectively. It is therefore impossible that an individual in DailyAggregate is *not* a GoalState, and therefore DailyAggregate $\sqsubseteq$ GoalState. □

More generally, in order to check subsumption $\hat{C} \sqsubseteq \hat{D}$, we can check satisfiability of its negation $\hat{C} \sqcap \neg\hat{D}$. This requires the negation of $\hat{D}$. Concept satisfiability w.r.t. acyclic TBox can be done without the TBox since we can unfold a concept that we want to check

$$
\begin{aligned}
\mathsf{StateA} \sqsubseteq\ & \exists\ \mathsf{exercise.Treadmill} \sqcap = .(\mathsf{id}, 1) \sqcap = .(\mathsf{HR}, 100) \sqcap \\
& = .(\mathsf{hours}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{hours})) \sqcap = .(\mathsf{calburn}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{calburn})) \sqcap \\
& = .(\mathsf{steps}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{steps})) \sqcap = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{steps}), 1) \sqcap \\
& = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{hours}), 1) \sqcap = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{calburn}), 1) \qquad (A6) \\
\mathsf{StateB} \sqsubseteq\ & \exists\ \mathsf{exercise.FlexStrider} \sqcap = .(\mathsf{id}, 2) \sqcap = .(\mathsf{HR}, 110) \sqcap \\
& = .(\mathsf{hours}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{hours})) \sqcap = .(\mathsf{calburn}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{calburn})) \sqcap \\
& = .(\mathsf{steps}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{steps})) \sqcap \\
& = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{hours}), 1) \sqcap = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{calburn}), 1) \sqcap \\
& = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{steps}), 1) \qquad (A7) \\
\mathsf{StateC} \sqsubseteq\ & \exists\ \mathsf{exercise.CrossTrainer} \sqcap = .(\mathsf{id}, 3) \sqcap = .(\mathsf{HR}, 128) \sqcap \\
& = .(\mathsf{hours}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{hours})) \sqcap = .(\mathsf{calburn}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{calburn})) \sqcap \\
& = .(\mathsf{steps}, \mathsf{sum}(\mathsf{exercise} \circ \mathsf{steps})) \sqcap \\
& = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{hours}), 1) \sqcap = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{calburn}), 1) \sqcap \\
& = .(\mathsf{count}(\mathsf{exercise} \circ \mathsf{steps}), 1) \qquad (A8) \\
\mathsf{DailyAggregate} \equiv\ & \exists\ \mathsf{has.StateA} \sqcap \exists\ \mathsf{has.StateB} \sqcap \exists\ \mathsf{has.StateC} \sqcap \\
& = .(\mathsf{hours}, \mathsf{sum}(\mathsf{has} \circ \mathsf{hours})) \sqcap = .(\mathsf{calburn}, \mathsf{sum}(\mathsf{has} \circ \mathsf{calburn})) \sqcap \\
& = .(\mathsf{steps}, \mathsf{sum}(\mathsf{has} \circ \mathsf{steps})) \sqcap = .(\mathsf{HR}, \mathsf{max}(\mathsf{has} \circ \mathsf{HR})) \sqcap \\
& = .(\mathsf{count}(\mathsf{has} \circ \mathsf{hours}), 3) \sqcap = .(\mathsf{count}(\mathsf{has} \circ \mathsf{calburn}), 3) \sqcap \\
& = .(\mathsf{count}(\mathsf{has} \circ \mathsf{steps}), 3) \sqcap = .(\mathsf{count}(\mathsf{has} \circ \mathsf{HR}), 3) \qquad (A9)
\end{aligned}
$$

Figure 6.5: A stream ontology recording daily fitness activities.

satisfiability using *unfolding* (Horrocks, 2003) to eliminate the TBox. Since $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ is not closed under negation, we can only perform *limited* subsumption checking, where $\hat{D}$ is restricted to formulas that do not contain any existential restriction (directly or indirectly), such as GoalState.

6.3.3 Consistency Checking

Given an $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ TBox $\mathcal{T}$, the *consistency checking* task is used to determine whether $\mathcal{T}$ is consistent. $\mathcal{T}$ is consistent if and only if there exists a model $\mathcal{I}$ for $\mathcal{T}$. If this is not the case, $\mathcal{T}$ is said to be inconsistent. An interpretation $\mathcal{I}$ is a model for a given $\mathcal{T}$ if and only if $\mathcal{I}$ satisfies all axioms in $\mathcal{T}$. Consistency checking can be reduced to concept satisfiability of $\top$. Let us use Example 6.3.3 to presents an inference problem, *consistency checking*.

Example 6.3.3. Given an example ontology in Figure 6.2, we can reason about this ontology by transforming all axioms to one axiom, where the concept $\top$ is on the left-hand side of that axiom as follow:

$$\begin{aligned}\top \sqsubseteq\ & (\neg \mathsf{RoadA} \sqcup = .(\mathsf{numberofcars}, 100)) \sqcap \\ & (\neg \mathsf{RoadB} \sqcup = .(\mathsf{numberofcars}, 200)) \sqcap \\ & (\neg \mathsf{RoadC} \sqcup = .(\mathsf{numberofcars}, 300)) \sqcap \\ & (\neg \mathsf{JammedRoad} \sqcup \geq .(\mathsf{numberofcars}, 500) \sqcap \\ & (\neg \mathsf{RoadD}\ \ \sqcup (\exists\ \mathsf{intersectWith.RoadA} \sqcap\ \exists\ \mathsf{intersectWith.RoadB} \sqcap \\ & \exists\ \mathsf{intersectWith.RoadC} \sqcap \leq .(\mathsf{numberofcars}, 600) \sqcap \\ & = .(\mathsf{numberofcars}, \mathsf{sum}(\mathsf{intersectWith} \circ \mathsf{numberofcars}))))\end{aligned}$$

Then we can check whether the concept $\top$ is satisfiable. Since our logic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ allows only acyclic TBoxes, it is sufficient to check the concept description on the left-hand side of this axiom. We need to calculate the sum of number of cars of each intersectWith-successors and check that the sum is less than or equal to 600. In this case, the intersectWith-successors belong to each concept RoadA, RoadB, and RoadC. Therefore, the sum of number of cars is 600 (100 + 200 + 300). Hence, the concept $\top$ is satisfiable and the example ontology in Figure 6.2 is consistency. □

6.4 Decidability of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$

Our logic is a restriction of the logic $\mathcal{EL}(\Sigma)$ introduced in Section 2.5.2, however we consider reasoning over ontologies with a TBox whereas $\mathcal{EL}(\Sigma)$ does not. The differences between our logic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ and $\mathcal{EL}(\Sigma)$ are described in Section 6.1. This section shows how the proof that concept satisfiability is decidable in (Baader & Sattler, 2003) can be adapted to $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ with TBox, under the restrictions introduced in Section 6.1.

The decision procedure in (Baader & Sattler, 2003) (referred to as $\mathcal{EL}(\Sigma)$-DP hereafter) has two steps. It first constructs (non-deterministically) an interpretation that is consistent with the abstract domain using completion rules, and then checks that this interpretation is consistent with the concrete domain as well. The main theorem states that this is a decision procedure if consistency of the concrete domain (i.e., Σ-consistency) is decidable. The proof relies on the fact that even though we may have to introduce additional $R \circ f$-successors when checking consistency of the concrete domain, we can always do so without them affecting the decidability of the abstract domain part, since the logic $\mathcal{EL}(\Sigma)$ does not permit universal quantification (Remark 16 in (Baader & Sattler, 2003)).

As mentioned in Section 2.5.2, since the logic $\mathcal{EL}(\Sigma)$ only supports concept satisfiability checking, Remark 16 from (Baader & Sattler, 2003) is sufficient only for aggregations count. If we add TBox to $\mathcal{EL}(\Sigma)$, Remark 16 is still sufficient for aggregation count even if TBox consistency checking is considered. However, it does not hold for the aggregations sum, min and max for TBox consistency checking. Intuitively, using negated concepts in a concept description, we can easily achieve the effect of universal restrictions on feature values without allowing universal restrictions in the logic itself. The aggregations sum, min and max may force the introduction of additional individuals with particular feature values, which may contradict the universally quantified value. Therefore, the $\mathcal{EL}(\Sigma)$-DP does not support $\mathcal{EL}(\Sigma)$ with TBoxes. We will show this through the following lemmas 6.4.1, 6.4.2, and 6.4.3.

Lemma 6.4.1. *$\mathcal{EL}(\Sigma)$-DP in (Baader & Sattler, 2003) does not work for TBox consistency checking, where the aggregation* count *is replaced by* sum *or the aggregation includes* sum.

Proof. We prove the lemma through a counterexample. The following TBox in 6.4.1 below will be considered satisfiable by $\mathcal{EL}(\Sigma)$-DP as it will check constraints in the abstract domain first, and then check the concrete domain, without switching back to abstract domain reasoning. In order to check TBox consistency of the following TBox in 6.4.1, we can easily check whether the concept $\top$ is satisfiable.

$$\top \sqsubseteq \underbrace{(A \sqcup P_{\geq 5}(f)) \sqcap (\neg A \sqcup P_{\geq 5}(f))}_{\text{(L1.1)}} \sqcap \underbrace{(\exists R.(P_{=5}(f)))}_{\text{(L1.2)}} \sqcap \underbrace{(P_{=6}(\mathsf{sum}(R \circ f)))}_{\text{(L1.3)}} \tag{6.4.1}$$

It is sufficient to check the left-hand side of axiom of TBox in 6.4.1. Concept description (L1.1) forces all f-values to be at least 5. Concept description (L1.2) states that there exists a R-successor with f-value equal to 5. Finally, concept description (L1.3) introduces an additional $R \circ f$-successor with f-value equal to 1 to satisfy the sum constraint (that the sum must be 6). This contradicts (L1.1), hence the concept $\top$ is unsatisfiable and TBox in 6.4.1 is inconsistent. However, $\mathcal{EL}(\Sigma)$-DP in (Baader & Sattler, 2003) is not able to identify this contradiction as it does not check the constraints on this new $R \circ f$-successor again (as it does not switch back to abstract domain reasoning). □

Lemma 6.4.2. *$\mathcal{EL}(\Sigma)$-DP in (Baader & Sattler, 2003) does not work for TBox consistency checking, where the aggregation includes* min*.*

Proof. We prove the lemma through a counterexample. The following TBox in 6.4.2 below will be considered satisfiable by $\mathcal{EL}(\Sigma)$-DP as it will check constraints in the abstract domain first, and then check the concrete domain, without switching back to abstract domain reasoning. In order to check TBox consistency of the following TBox in 6.4.2, we can easily check whether the concept $\top$ is satisfiable.

$$\top \sqsubseteq \underbrace{(A \sqcup P_{\geq 5}(f)) \sqcap (\neg A \sqcup P_{\geq 5}(f))}_{\text{(L1.1)}} \sqcap \underbrace{(\exists R.(P_{=5}(f)))}_{\text{(L1.2)}} \sqcap \underbrace{(P_{=4}(\mathsf{min}(R \circ f)))}_{\text{(L1.3)}} \tag{6.4.2}$$

It is sufficient to check the left-hand side of axiom of TBox in 6.4.2.Concept description (L1.1) forces all f-values to be at least 5. Concept description (L1.2) states that there exists a R-successor with f-value equal to 5. Finally, concept description (L1.3) introduces an additional $R \circ f$-successor with f value equal to 4 to satisfy the min constraint (that the minimum must be 4). This contradicts (L1.1), hence the concept $\top$ is unsatisfiable and TBox in 6.4.2 is inconsistent. However, $\mathcal{EL}(\Sigma)$-DP in (Baader & Sattler, 2003) is not able to identify this contradiction as it does not check the constraints on this new $R \circ f$-successor again (as it does not switch back to abstract domain reasoning). □

Lemma 6.4.3. *$\mathcal{EL}(\Sigma)$-DP in (Baader & Sattler, 2003) does not work for TBox consistency checking, where the aggregation includes* max*.*

Proof. We prove the lemma through a counterexample. The following TBox in 6.4.3 below will be considered satisfiable by $\mathcal{EL}(\Sigma)$-DP as it will check constraints in the abstract domain first, and then check the concrete domain, without switching back to abstract domain reasoning. In order to check TBox consistency of the following TBox in 6.4.3, we can easily check whether the concept $\top$ is satisfiable.

$$\top \sqsubseteq \underbrace{(A \sqcup P_{\leq 5}(f)) \sqcap (\neg A \sqcup P_{\leq 5}(f))}_{\text{(L1.1)}} \sqcap \underbrace{(\exists R.(P_{=5}(f)))}_{\text{(L1.2)}} \sqcap \underbrace{(P_{=6}(\mathsf{max}(R \circ f)))}_{\text{(L1.3)}} \tag{6.4.3}$$

It is sufficient to check the left-hand side of axiom of TBox in 6.4.3. Concept description (L1.1) forces all f-values to be at most 5. Concept description (L1.2) states that there exists a R-successor with f-value equal to 5. Finally, concept description (L1.3) introduces an additional $R \circ f$-successor with f-value equal to 6 to satisfy the max constraint (that the maximum must be 6). This contradicts (L1.1), hence the concept $\top$ is unsatisfiable and TBox in 6.4.3 is inconsistent. However, $\mathcal{EL}(\Sigma)$-DP in (Baader & Sattler, 2003) is not able to identify this contradiction as it does not check the constraints on this new $R \circ f$-successor again (as it does not switch back to abstract domain reasoning). □

With the axiom syntax defined in Table 6.2, negated concepts are not allowed on the left hand side of subclass axioms in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. Therefore, the effect of universal restriction does not manifest here. That way, new $R \circ f$-successors can be safely added outside any named concepts, where they cannot affect satisfiability of any of the axioms of abstract domain. What remains to be shown is that consistency of our concrete domain, which includes aggregations min, max, count, sum w.r.t. acyclic TBox, is decidable.

Before we show the decidability proof, we will introduce the proof technique from (Baader & Sattler, 2003). Since the behaviour of aggregation functions is axiomatised, aggregation variables $\Gamma(Y)$ can be replaced by individual variables y_Γ. For $\Delta^{\mathbb{N}}$ the set of natural numbers, the relational operators $\geq, <, \leq, >, =,$ and $\neq$ are defined as usual. The constraint system in $\mathcal{EL}(\Sigma)$-DP (Baader & Sattler, 2003) is generated with the conjunction of $\mathbb{N}$-constraints. Since the proof is for $\mathbb{N}$-consistency, which is a special case of Σ–consistency where the concrete domain is natural numbers, the conjunction of $\mathbb{N}$-constraints is transformed to the set of linear (in)equalities. Then the Boolean combination of linear inequalities is added as constraints in order to handle interactions between aggregations sum, count, max, and min.

Lemma 6.4.4. *If $\mathbb{N}$ is a concrete domain such that*

- $\Delta^{\mathbb{N}}$ *is the set of natural numbers*
- $\bowtie(\mathbb{N}) = \{\geq, <, \leq, >, =, \neq\}$
- $\Sigma(\mathbb{N}) = \{min, max, count, sum\}$

with Restriction 3 in Section 6.1, then $\mathbb{N}$-consistency is decidable.

Proof. This proof is adapted from the proof of lemma 24 in (Baader & Sattler, 2003). The approach for proving decidability in (Baader & Sattler, 2003) is to transform a constraint system into an equisatisfiable set of linear (in)equalities without aggregation functions. Aggregation functions are replaced by newly introduced variables plus additional (in)equalities that implement consistency constraints between the new variables. The only difference is that aggregated multiset variables involving sum are replaced by individual variable y_{sum}. Let $\mathcal{A}$ be a constraint system, $\mathcal{A}_{\mathbb{N}}$ be the conjunction of $\mathbb{N}$-constraints in $\mathcal{A}$, σ be a set of multiset variables, Y be a multiset. Let $D_{\mathcal{A}}$ be a set of linear (in)equalities without aggregation that is satisfiable iff $\mathcal{A}_{\mathbb{N}}$ is satisfiable. In order to handle sum, the Boolean combination of linear inequalities $D_{y_{\mathsf{sum}}}$ are added to $D_{\mathcal{A}}$ for aggregated multiset $\mathsf{sum}(Y)$. Every $\mathsf{max}(Y)$, $\mathsf{min}(Y)$, $\mathsf{count}(Y)$, and $\mathsf{sum}(Y)$ is replaced by y_{max}, y_{min}, y_{count} and y_{sum} respectively.

Let a set $D_{\mathcal{A}}$ be obtained from $D'_{\mathcal{A}}$ by replacing constraints by (in)equalities. Precisely, $D'_{\mathcal{A}}$ is the set of all concrete predicates $\bowtie(d_1, d_2)$, where d_1, d_2 are feature values, in $\mathcal{A}_{\mathbb{N}}$ where each occurrence of $\mathsf{max}(Y)$, $\mathsf{min}(Y)$, $\mathsf{count}(Y)$, and $\mathsf{sum}(Y)$ is replaced by y_{max}, y_{min}, y_{count} and y_{sum} respectively. Then $D_{\mathcal{A}}$ is obtained from $D'_{\mathcal{A}}$ by replacing constraints with (in)equalities and adding axioms to handle the interactions between $\mathsf{max}(Y)$, $\mathsf{min}(Y)$, $\mathsf{count}(Y)$, and $z : Y$ fro each Y. The sets $D'_{\mathcal{A}}$ and $D_{\mathcal{A}}$ is defined as follows:

$$\begin{aligned}
D'_{\mathcal{A}} := {} & \mathcal{A}_{\mathbb{N}}[\mathsf{max}(Y)/y_{\mathsf{max}} \text{ for } Y \in \sigma][\mathsf{min}(Y)/y_{\mathsf{min}} \text{ for } Y \in \sigma][\mathsf{count}(Y)/y_{\mathsf{count}} \text{ for } Y \in \sigma] \\
& [\mathsf{sum}(Y)/y_{\mathsf{sum}} \text{ for } Y \in \sigma] \\
D_{\mathcal{A}} := {} & \{y_{\mathsf{min}} \le y_{\mathsf{max}} | y_{\mathsf{min}} \text{ or } y_{\mathsf{max}} \text{ occurs in } D'_{\mathcal{A}}\} \cup \\
& \{y_{\mathsf{min}} \le z | y_{\mathsf{min}} \text{ occurs in } D'_{\mathcal{A}} \text{ and } (z : Y) \in \mathcal{A}_{\mathbb{N}}\} \cup \\
& \{y_{\mathsf{max}} \ge z | y_{\mathsf{max}} \text{ occurs in } D'_{\mathcal{A}} \text{ and } (z : Y) \in \mathcal{A}_{\mathbb{N}}\} \cup \\
& \{x \bowtie y | \bowtie \in \{>, \ge, <, \le, =, \ne\} \text{ and } \bowtie .(x, y) \in D'_{\mathcal{A}}\} \cup \\
& \{D_{y_{\mathsf{count}}} | y_{\mathsf{count}} \text{ occurs in } D'_{\mathcal{A}}\}
\end{aligned}$$

Let $x_Y := \{x | x : Y \in \mathcal{C}\}$, x_Y^{count} be the cardinality of x_Y, and x_Y^{sum} be the sum of x_Y. Then $D_{y_{\mathsf{count}}}$ is defined as follows:

$$\begin{aligned}
D_{y_{\mathsf{count}}} := & \Bigg(\Big(x_Y^{\mathsf{count}} = y_{\mathsf{count}} \wedge \bigvee_{x \in x_Y} x = y_{\mathsf{min}} \wedge \bigvee_{x \in x_Y} x = y_{\mathsf{max}} \Big) \vee && \text{(DC1)} \\
& \Big(x_Y^{\mathsf{count}} = y_{\mathsf{count}} - 1 \wedge \bigvee_{x \in x_Y} (x = y_{\mathsf{min}} \vee x = y_{\mathsf{max}}) \Big) \vee && \text{(DC2)} \\
& \Big(x_Y^{\mathsf{count}} \le y_{\mathsf{count}} - 2 \Big) \Bigg) \wedge && \text{(DC3)} \\
& y_{\mathsf{count}} \in \mathbb{N} \wedge y_{\mathsf{count}} \ge 0
\end{aligned}$$

The disjunctions above are important since we need to consider the presence of minimum and maximum values in the multiset. The first disjunct (DC1) handles the case, where both minimum and maximum values are in the multiset. The second disjunct (DC2) handles the case, where either minimum or maximum value is not in the multiset. Therefore, the result of aggregation count is the number of elements in the multiset plus one. The third disjunct (DC3) handles the case where both minimum and maximum values are not in the multiset. As a consequence, the result of aggregation count is the number of elements in the multiset plus two. In order to handle the interactions between $\mathsf{max}(Y)$, $\mathsf{min}(Y)$, $\mathsf{count}(Y)$, and $\mathsf{sum}(Y)$, $D_{\mathcal{A}}^{\mathsf{sum}}$ is obtained by replacing boolean combination $D_{y_{\mathsf{sum}}}$ for each y_{sum} in $D_{\mathcal{A}}$. Then $D_{\mathcal{A}}^{\mathsf{sum}}$ is defined as follows:

$$D_{\mathcal{A}}^{\mathsf{sum}} := D_{\mathcal{A}} \cup \bigcup_{y_{\mathsf{sum}} \text{ occurs in } D_{\mathcal{A}}} D_{y_{\mathsf{sum}}},$$

Then $D_{y_{\mathsf{sum}}}$ is defined as follows:

$$\begin{aligned}
D_{y_{\mathsf{sum}}} := & \, y_{\mathsf{sum}} \ge y_{\mathsf{min}} \times y_{\mathsf{count}} \wedge \\
& y_{\mathsf{sum}} \le y_{\mathsf{max}} \times y_{\mathsf{count}} \wedge \\
& \Big(\big(x_Y^{\mathsf{sum}} = y_{\mathsf{sum}} \wedge \bigvee_{x \in x_Y} x = y_{\mathsf{min}} \wedge \bigvee_{x \in x_Y} x = y_{\mathsf{max}} \big) \vee && \text{(DS1)} \\
& \big(x_Y^{\mathsf{sum}} = y_{\mathsf{sum}} - y_{\mathsf{max}} \wedge \bigvee_{x \in x_Y} x = y_{\mathsf{min}} \wedge y_{\mathsf{max}} \notin x_Y \big) \vee && \text{(DS2)} \\
& \big(x_Y^{\mathsf{sum}} = y_{\mathsf{sum}} - y_{\mathsf{min}} \wedge \bigvee_{x \in x_Y} x = y_{\mathsf{max}} \wedge y_{\mathsf{min}} \notin x_Y \big) \vee && \text{(DS3)} \\
& \big(x_Y^{\mathsf{sum}} \le y_{\mathsf{sum}} - y_{\mathsf{max}} - y_{\mathsf{min}} \big) \Big) \wedge && \text{(DS4)} \\
& y_{\mathsf{sum}} \in \mathbb{N}
\end{aligned}$$

Similar to the aggregation count, the disjunctions are used to handle a presence of minimum and maximum values in the multiset. The first disjunct (DS1) handles the case, where both minimum and maximum values are in the multiset. The second disjunct (DS2) handles the case, where the maximum value is not in the multiset. Therefore, the result of aggregation sum is the sum of elements in the multiset plus the maximum value. The third disjunct (DS3) handles the case, where the minimum value is not in the multiset. Therefore, the result of aggregation sum is the sum of elements in the multiset plus the minimum value. The forth disjunct (DS4) handles the case, where both minimum and maximum values are not in the multiset. As a consequence, the result of aggregation sum is the sum of elements in the multiset plus both minimum and maximum values.

This conjunction of $y_{\mathsf{sum}} \geq y_{\mathsf{min}} \times y_{\mathsf{count}}$ and $y_{\mathsf{sum}} \leq y_{\mathsf{max}} \times y_{\mathsf{count}}$ is necessary because it is used to handle the interactions between all aggregations. Intuitively, the average of each element in Y needs to be between the min value and the max value. The first conjunct ($y_{\mathsf{sum}} \geq y_{\mathsf{min}} \times y_{\mathsf{count}}$) ensures that the average of each element in Y is more than or equal to min value. The second conjunct ($y_{\mathsf{sum}} \leq y_{\mathsf{max}} \times y_{\mathsf{count}}$) ensures that the average of each element in Y is less than or equal to max value. Finally, since the concrete domain is always finite due to Restriction 3, the satisfiability of $D_{\mathcal{A}}^{\mathsf{sum}}$ can be decided using linear, integer or mixed programming techniques. $\square$

6.5 Related Works

Concrete domains in Description Logics represent concrete qualities, such as size, weight, temperature, via *feature*. Concrete domains have first been proposed as an extension of $\mathcal{ALC}$ in $\mathcal{ALC}(\mathcal{D})$ (Baader & Hanschke, 1991). $\mathcal{ALC}(\mathcal{D})$ allows *concrete feature*, which relates (abstract) individuals to concrete values and *abstract feature*, which relates individuals to each other in the abstract domain. It also supports feature chains (e.g., $f_1 \cdots f_n$). It has been shown (Lutz, 2002) that the complexity of concept satisfiability and subsumption of $\mathcal{ALC}(\mathcal{D})$ is PSapce-complete. Concrete domains have since been further developed in several directions.

The Description Logic $\mathcal{ALCF}(\mathcal{D})$ extends $\mathcal{ALC}(\mathcal{D})$ by adding *feature agreements* and *feature disagreements* (Lutz, 2002). The complexity of concept satisfiability and subsumption checking in $\mathcal{ALCF}(\mathcal{D})$ is PSapce-complete. *Functional dependencies* in $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ (Lutz & Milicic, 2004) allows one to express that a set of properties can decide the value of a property. For example, all employees with the same department id work in the same department. $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ is in general undecidable. Decidability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ is retained by allowing only *safe key boxes*. The complexity of concept satisfiability of $\mathcal{ALC}(\mathcal{D})^{\mathcal{FD}}$ with safe key boxes is NExpTime-hard. Note that the complexity depends on concrete doamain.

Aggregation over concrete domains was originally proposed in $\mathcal{ALC}(\Sigma)$ (Baader & Sattler, 2003). $\mathcal{ALC}(\Sigma)$ allows all $\mathcal{ALC}$ concept constructs and aggregations. However, this leads to undecidability of concept satisfiability and subsumption. Decidability of concept satisfiability is retained by disallowing universal quantification and restricting the use of negation to named concepts for certain concrete domains and aggregations, resulting in the logic $\mathcal{EL}(\Sigma)$ (Baader & Sattler, 2003). A set of completion rules was proposed for concept satisfiability of $\mathcal{EL}(\Sigma)$, for reasoning on the abstract domain and for creating conjunctions of aggregations, features, and predicates as constraints (concrete domain reasoning). These conjunctions can be subsequently decided by techniques such as linear or mixed integer programming. However, no reasoning support has been implemented for this logic. The DL $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ is a restriction of $\mathcal{EL}(\Sigma)$ (Baader & Sattler, 2003). On the other hand, the DL $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ is equiped with TBox and aggregation *sum*, which allow us to express interesting knowledge such as the example in Figure 6.2. In this work we also show that our logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ is decidable for a certain concrete domain. Our logic is designed to allow us to perform reasoning tasks that require a TBox such as *ontology consistency* and *concept subsumption checking* in addition to concept satisfiability.

Chapter 7

Encoding $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ into MiniZinc

As described in Chapter 6, extending Description Logics with *concrete domains* with *aggregations*, such as sums, counting, or min/max, allows us to model ontologies that are useful and interesting in many domains (we used the fitness tracker example in Chapter 6). Aggregation is a very natural extension, since it allows individuals to be organised by their physical attribute values. However, reasoning support for aggregations has not received much attention, and proves challenging. Even though one tableau-based algorithm was proposed to support concept satisfiability (see Section 3.1.2), it has not been implemented. Therefore, it is difficult to show the effectiveness of this algorithm.

In Chapter 4, we show that Constraint Programming (CP) offers powerful search and modelling techniques. On the one hand, modelling languages such as MiniZinc support modelling for many types of constraints, including numerical constraint. In addition, CP offers modelling techniques such as symmetry breaking, which can reduce search during the solving process. On the other hand, CP provides mature solving techniques. CP can highly prune the search tree by maintaining consistencies during the search – this technique is called Constraint Propagation. CP is also good at handling numerical constraints. Moreover, recent advanced CP techniques such as Lazy Clause Generation can combine the advantages of CP techniques and SAT techniques, resulting in powerful learning solvers. Constraint Programming is also an active research area, resulting in new solvers and techniques being continuously proposed with ongoing improvements in performance.

Chapter 5 has shown that it is possible to efficiently perform reasoning in Description Logic (i.e., $\mathcal{ALC}$) by encoding it into a CP problem (i.e., constraint model). Particularly, it has been shown that CP techniques are able to deal with one of the sources of inefficiency of tableau-based algorithms, OR-search. Considering the experiences gained and results of the evaluations in Chapter 5, in this chapter we show that CP techniques and solvers can be very powerful and suitable tools to reason about Description Logic with concrete domain and aggregations. In our case, we consider reasoning support for $\mathcal{ELU}^{(\neg)}(f, \Sigma)$.

We start our research from an encoding of concept satisfiability in $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ w.r.t. acyclic TBoxes into MiniZinc models. Then we further investigate this idea for TBox consistency checking and limited concept subsumption checking (see Section 6.3). We chose to deal with only acyclic TBoxes and postponed the issue of introducing blocking techniques to handle cyclicity to future work.

The rest of this chapter is structured as follows. In Section 7.1, we define our novel encoding scheme to encode $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ into CP problems. Then we prove that our encoding scheme is sound and complete, and show the complexity of the resulting reasoning problems in $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ in Section 7.2. Next, in Section 7.3, we discuss symmetry breaking and search heuristic in MiniZinc constraint models, which are important optimisation techniques that improve the performance of our approach. Finally, we close this chapter with an empirical evaluation, describing the setup and discussing the results in Section 7.4.

7.1 Concept Satisfiability and Limited Subsumption Checking

In this section, we extend our previous encoding of $\mathcal{ALC}$ into MiniZinc in order to support reasoning over concrete domain and aggregations in $\mathcal{ELU}^{(\neg)}(f, \Sigma)$.

7.1.1 $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ to MiniZinc

This section presents the close relation of MiniZinc syntax with the semantics of $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ as defined in Table 6.1.

Let us recall some basic definitions. *Individuals* are encoded as positive integers. Moreover, we encode the top concept $\top$ (the superclass of all concepts) and a concept A as set variables of individuals `T` and `A` respectively. `T` is a set of all abstract individuals in encoding. Hence, `A` is always a subset of `T` for all concepts A. The bottom concept $\bot$ can be encoded to the empty set (`{}`). A role R is encoded as an array of sets `R`, where indices `i` are individuals in one set and the set `R[i]` contains R-successors of individual `i`.

Let us now look at the new syntactic constructs in $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. A (functional) feature f maps an abstract individual to a natural number. It is encoded as an array of natural numbers `f`, whose indices `i` are individuals in its domain, and the natural number `f[i]` is the f-value (f-successor) of individual `i`.

In Sections 4.2.2 and 5.1, the MiniZinc encodings for the Description Logics $\mathcal{EL}$ and $\mathcal{ALC}$ are presented. $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ allows concept name, conjunctions, disjunctions, atomic negations, and existential restrictions, which are the same as language constructs in $\mathcal{ALC}$. The only one language construct left to be explained is *concrete domain* concept construct. This concept construct can be encoded as follows:

Concrete domain ($\bowtie .(F_1, F_2)$) is the set of individuals that have F_1-successor and F_2-successor, and F_1-successor and F_2-successor satisfy $\bowtie$. Therefore, it can be encoded to $\mathtt{F}_1 \bowtie \mathtt{F}_2$, where $\bowtie \in \{\geq, <, \leq, >, =, \neq\}$, and expressions `F` (such as $\mathtt{F}_1$ and $\mathtt{F}_2$) are encoded as the following three cases:

- If `F` is an atomic feature f, it is encoded as an array of natural numbers `f[i]`, whose indices `i` are individuals in its domain, and the natural number `f[i]` is the f-value (f-successor) of individual `i`.
- If `F` is a natural number d, it is encoded as natural number `d`.
- If `F` is an aggregation $\Gamma(R \circ f)$, where $\Gamma \in \{\mathsf{sum}, \mathsf{count}, \mathsf{min}, \mathsf{max}\}$, the basic idea is to encode it as an aggregation over $R \circ f$-successors. For example, $\mathsf{sum}(R \circ f)$ is translated into an expression similar to `sum(j in R[i])(f[j])`. The full encoding of aggregations is more complex and will be explained below.

Let us now use an example to illustrate the encoding of concrete domain constructs. Given a concept $= .(\mathsf{weight}, \mathsf{sum}(\mathsf{has} \circ \mathsf{weight}))$, this concept defines a set of individuals whose weight is equal to the sum of the weights of their has-successors. It can be encoded into `weight[i]` $=$ `sum(j in has[i])(weight[j])`, where `i` and `j` are individuals.

We can now formally define the encoding rules of $\mathcal{ELU}^{(\neg)}(f, \Sigma)$.

Definition 7.1.1. $\mathcal{ELU}^{(\neg)}(f, \Sigma)$2MiniZinc (MiniZinc Encoding : Set-based Encoding) Let $\mathcal{T}$ be an acyclic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ TBox in normal form. The encoding scheme uses the following notation:

- $\mathtt{T}$ is a set of all abstract individuals of $\mathcal{T}$ ($\Delta^{\mathcal{I}}$)
- $\mathtt{A}$ is a set of abstract individuals of a concept A
- $\mathtt{R}$ is an array of sets of abstract individuals that are related via R
- $\mathtt{f}$ is an array of concrete individuals that are related via f
- A and B are atomic concepts
- i, j are abstract individuals
- n, m, k are non-negative integers
- $\mathtt{x}_{\Gamma}^{R \circ f}$ is an array of natural numbers of each aggregation, where $\Gamma \in \{\mathsf{sum}, \mathsf{count}, \mathsf{max}, \mathsf{min}\}$
- $\mathtt{sx}_{\Gamma}^{R \circ f}$ is an array of natural numbers of each aggregation count and sum, where $\Gamma \in \{\mathsf{sum}, \mathsf{count}\}$
- $\mathtt{bx}_{\Gamma}^{R \circ f}$ is an array of boolean variables for each aggregation max and min, where $\Gamma \in \{\mathsf{max}, \mathsf{min}\}$. The value for each array index will be true if the value of $\mathtt{x}_{\Gamma}^{R \circ f}$ is in its multiset. Otherwise, the value will be false.
- **`bool2int`** is the MiniZinc function to convert Booleans to integers, i.e., true = 1 and false = 0
- **`ub_array`** is the MiniZinc function to get the maximum of all upper bounds of the elements in an array
- $\varphi^{\mathcal{T}}$ is a set of constraints

The encoding scheme is defined with the following rules:

ELU-ER1 Initialisation: for every R and f, introducing $\mathtt{x}_{\Gamma}^{R \circ f}$, $\mathtt{sx}_{\Gamma}^{R \circ f}$, $\mathtt{bx}_{\Gamma}^{R \circ f}$, and

$$\begin{aligned}(\forall i \in \mathtt{T} : \mathtt{x}_{\mathtt{count}}^{R \circ f}\mathtt{[}i\mathtt{]} = &\mathbf{card}(\mathtt{R[}i\mathtt{]}) + \mathtt{sx}_{\mathtt{count}}^{R \circ f}\mathtt{[}i\mathtt{]} + \mathbf{bool2int}(\mathtt{bx}_{\mathtt{max}}^{R \circ f}\mathtt{[}i\mathtt{]}) \\ &+ \mathbf{bool2int}(\mathtt{bx}_{\mathtt{min}}^{R \circ f}\mathtt{[}i\mathtt{]})) \in \varphi^{\mathcal{T}}\end{aligned} \tag{7.1.1}$$

$$\begin{aligned}(\forall i \in \mathtt{T} : \mathtt{x}_{\mathtt{sum}}^{R \circ f}\mathtt{[}i\mathtt{]} = &\mathbf{sum}(j \in \mathtt{R[}i\mathtt{]})(\mathbf{if}\ j \in \mathtt{R[}i\mathtt{]}\ \mathbf{then}\ \mathtt{f[}j\mathtt{]}\ \mathbf{else}\ 0\ \mathbf{endif}) \\ &+ \mathtt{sx}_{\mathtt{sum}}^{R \circ f}\mathtt{[}i\mathtt{]}) \in \varphi^{\mathcal{T}}\end{aligned} \tag{7.1.2}$$

$$\begin{aligned}&(\forall i \in \mathtt{T} : (\mathtt{x}_{\mathtt{min}}^{R \circ f}\mathtt{[}i\mathtt{]} = \mathbf{min}(j \in \mathtt{R[}i\mathtt{]})(\mathbf{if}\ j \in \mathtt{R[}i\mathtt{]}\ \mathbf{then}\ \mathtt{f[}j\mathtt{]}\ \mathbf{else}\ \mathbf{ub\_array}(\mathtt{f}) \\ &+ 1\ \mathbf{endif})) \lor ((\mathtt{x}_{\mathtt{min}}^{R \circ f}\mathtt{[}i\mathtt{]} < \mathbf{min}(j \in \mathtt{R[}i\mathtt{]})(\mathbf{if}\ j \in \mathtt{R[}i\mathtt{]}\ \mathbf{then}\ \mathtt{f[}j\mathtt{]}\ \mathbf{else}\ \mathbf{ub\_array}(\mathtt{f}) \\ &+ 1\ \mathbf{endif})) \land \mathtt{bx}_{\mathsf{min}}^{R \circ f}\mathtt{[}i\mathtt{]})) \in \varphi^{\mathcal{T}}\end{aligned} \tag{7.1.3}$$

$$\begin{aligned}&(\forall i \in \mathtt{T} : (\mathtt{x}_{\mathtt{max}}^{R \circ f}\mathtt{[}i\mathtt{]} = \mathbf{max}(j \in \mathtt{R[}i\mathtt{]})(\mathbf{if}\ j \in \mathtt{R[}i\mathtt{]}\ \mathbf{then}\ \mathtt{f[}j\mathtt{]}\ \mathbf{else}\ 0\ \mathbf{endif})) \lor \\ &((\mathtt{x}_{\mathtt{max}}^{R \circ f}\mathtt{[}i\mathtt{]} > \mathbf{max}(j \in \mathtt{R[}i\mathtt{]})(\mathbf{if}\ j \in \mathtt{R[}i\mathtt{]}\ \mathbf{then}\ \mathtt{f[}j\mathtt{]}\ \mathbf{else}\ 0\ \mathbf{endif})) \land \mathtt{bx}_{\mathsf{min}}^{R \circ f}\mathtt{[}i\mathtt{]})) \in \varphi^{\mathcal{T}}\end{aligned} \tag{7.1.4}$$

$$(\forall i \in \mathtt{T} : \mathtt{sx}_{\mathtt{sum}}^{R \circ f}\mathtt{[}i\mathtt{]} \geq \mathtt{x}_{\mathtt{min}}^{R \circ f}\mathtt{[}i\mathtt{]} \times \mathtt{sx}_{\mathsf{count}}^{R \circ f}\mathtt{[}i\mathtt{]}) \in \varphi^{\mathcal{T}} \tag{7.1.5}$$

$$(\forall i \in \texttt{T} : \texttt{sx}^{R\circ f}_{\texttt{sum}}\texttt{[}i\texttt{]} \leq \texttt{x}^{R\circ f}_{\texttt{max}}\texttt{[}i\texttt{]} \times \texttt{sx}^{R\circ f}_{\texttt{count}}\texttt{[}i\texttt{]}) \in \varphi^{\mathcal{T}} \tag{7.1.6}$$

$$(\forall i \in \texttt{T} : \texttt{x}^{R\circ f}_{\texttt{min}}\texttt{[}i\texttt{]} \leq \texttt{x}^{R\circ f}_{\texttt{max}}\texttt{[}i\texttt{]}) \in \varphi^{\mathcal{T}} \tag{7.1.7}$$

ELU-ER2 For every axiom $A \sqsubseteq \sqcup_m B_m \in \mathcal{T}$,

$$(\texttt{A} \subseteq (\bigcup_m \texttt{B}_m)) \in \varphi^{\mathcal{T}} \tag{7.1.8}$$

ELU-ER3 For every axiom $A \sqsubseteq \exists R.B \in \mathcal{T}$,

$$\texttt{A} \subseteq \{i \mid \forall i \in \texttt{T} \wedge \texttt{card(R[}i\texttt{]} \cap \texttt{B)} \geq 1\} \in \varphi^{\mathcal{T}} \tag{7.1.9}$$

ELU-ER4 For every axiom $A \equiv \sqcap_m B_m \in \mathcal{T}$,

$$(\texttt{A} = (\bigcap_m \texttt{B}_m)) \in \varphi^{\mathcal{T}} \tag{7.1.10}$$

ELU-ER5 For every axiom $A \sqsubseteq \bowtie .(F_1, F_2) \in \mathcal{T}$,

$$\texttt{A} \subseteq \{i \mid \forall i \in \texttt{T} \wedge \mathsf{tran}(F_1) \bowtie \mathsf{tran}(F_2)\} \in \varphi^{\mathcal{T}} \text{ , where } \bowtie \in \{\geq, <, \leq, >, =, \neq\} \tag{7.1.11}$$

where

- If $F_k = f$, then
$$\mathsf{tran}(F_k) = \texttt{f[}i\texttt{]}$$
- If $F_k = d$ (constant natural number), then
$$\mathsf{tran}(F_k) = \texttt{d}$$
- If $F_k = \Sigma(R \circ f)$, then
$$\mathsf{tran}(F_k) = \texttt{x}^{R\circ f}_{\Gamma}\texttt{[}i\texttt{]}$$
where $\texttt{x}^{R\circ f}_{\Gamma}$ is an array of natural numbers

ELU-ER6 For the concept name A for which we want to check satisfiability,

$$(\texttt{A} \neq \emptyset) \in \varphi^{\mathcal{T}} \tag{7.1.12}$$

Note that a concept definition can be easily encoded by changing $\subseteq$ to $=$. A negated concept name ($\neg B$) can be translated to (`T - B`), where ”`-`” is set difference.

Next, we present the intuition of our encoding. This encoding directly follows the semantics of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ in Table 6.1. We assume that an $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox $\mathcal{T}$ in negation normal form contains only axioms of the forms: (1) $A \sqsubseteq B$, (2) $A \sqsubseteq \sqcup_j B_j$, (3) $A \sqsubseteq \exists R.B$, (4) $A \sqsubseteq \bowtie .(F_1, F_2)$, (5) $A \equiv \sqcap_i B_i$, (6) $A \equiv \sqcup_i B_i$, (7) $A \equiv \exists R.B$, and (8) $A \equiv \bowtie .(F_1, F_2)$. Let `i` be an individual, n, m, k be non-negative integers, and $\varphi^{\mathcal{T}}$ be a set of MiniZinc constraints. Axioms in $\mathcal{T}$ are then encoded by executing the above rules as follows, building up the set of constraints $\varphi^{\mathcal{T}}$:

- For every axiom $A \sqsubseteq \sqcup_m B_m$, rule **ELU-ER2** is applied to obtain a constraint of type (7.1.8). This rule is straightforward since the disjunction of concept names can be easily translated into a union operation of sets $\texttt{B}_i$, and the subclass relation $\sqsubseteq$ can be translated into a subset constraint in MiniZinc.

- For every axiom $A \sqsubseteq \exists R.B$, rule **ELU-ER3** is applied to obtain a constraint of type (7.1.9). The subsumption $\sqsubseteq$ is encoded into a subset operation. This constraint means that `A` is a subset of a set of individuals i such that $\forall i \in \Delta^{\mathcal{I}}$ and the cardinality of the intersection of R-successors of i and `B` is greater than 1. As can be seen, the meaning of this constraint is very close to the semantics of $\exists R.B$ in Table 6.1.

- For every axiom $A \equiv \sqcap_m B_m$, rule **ELU-ER4** is applied to obtain a constraint of type (7.1.10). This rule is straightforward since a conjunction of concept names can be easily translated into an intersection operation of sets $\texttt{B}_i$, and the equivalent relation $\equiv$ can be translated into the `=` constraint in MiniZinc.

In the following we describe the encoding rules for **ELU-ER1** and **ELU-ER5**, which involve concrete domains, features, and aggregations.

- For every role R, feature f, and aggregation $\Gamma \in \{\mathsf{sum}, \mathsf{count}, \mathsf{max}, \mathsf{min}\}$, rule **ELU-ER1** is applied to obtain constraints of type (7.1.1)-(7.1.7).

- For every axiom $A \sqsubseteq \bowtie .(F_1, F_2)$, rule **ELU-ER5** is applied to obtain a constraint of type (7.1.11). $\bowtie \in \{\geq, <, \leq, >, =, \neq\}$, and the function $\texttt{tran(}F_k\texttt{)}$ is defined as follows

$$\texttt{tran(}F_k\texttt{)} = \begin{cases} \texttt{f[i]} & \text{(i) if } F_k = f \text{ } (F_k \text{ is a feature}) \\ \texttt{d} & \text{(ii) if } F_k = d \text{ } (F_k \text{ is an integer value}) \\ \texttt{x}_{\Gamma}^{R\circ f}\texttt{[}i\texttt{]} & \text{(iii) if } F_k = \Gamma(R \circ f) \text{ } (F_k \text{ is an aggregation}) \end{cases}$$

Let us consider each case of aggregation as follows:

(i) A functional feature f is translated into an array mapping abstract individuals to integers `f`. Therefore, $\texttt{tran(}F\texttt{)}$ is encoded into `f[i]`.

(ii) A natural number d is directly translated into a natural number. Therefore, $\texttt{tran(}F\texttt{)}$ is encoded into `d` such that `d` is a natural number.

(iii) An aggregation $\Gamma(R \circ f)$ is translated into the corresponding MiniZinc array. The array aggregates (with aggregation function $\Gamma \in \{\mathsf{sum}, \mathsf{count}, \mathsf{max}, \mathsf{min}\}$) the feature values of the abstract domain individuals (indices of the array). Each aggregation function is translated directly to MiniZinc arithmetic operations **`sum`**, **`max`**, **`min`**, and set operation **`card`** (for count).

However, the aggregation over abstract domain R-successors may not be enough, and sometimes additional $R \circ f$-successors need to be introduced to satisfy the concrete domain constraints. To account for this additional contribution to an aggregation, *slack variables*, encoded as $\texttt{sx}_{\Gamma}^{R\circ f}\texttt{[}i\texttt{]}$, are introduced to hold the aggregated value of all these additional successors, without actually introducing abstract individuals. A slack variable thereby corresponds to the slack value that is left in the concrete domain constraints introduced by (Baader & Sattler, 2003).

For case (iii) where $\texttt{tran(}F\texttt{)}$ is encoded into $\texttt{x}_{\Gamma}^{R\circ f}\texttt{[}i\texttt{]}$, we next consider each aggregation as follows.

count: It is encoded into a constraint 7.1.1. The first part of the constraint $\texttt{card(R[}i\texttt{])}$ counts actual elements in the multiset of abstract individuals. $\texttt{sx}_{\mathsf{count}}^{R\circ f}\texttt{[}i\texttt{]}$ is a number of additional $R \circ f$-successors (i.e., slack) to satisfy the concrete domain constraints. $\texttt{bx}_{\mathsf{min}}^{R\circ f}\texttt{[}i\texttt{]}$ (resp. $\texttt{bx}_{\mathsf{max}}^{R\circ f}\texttt{[}i\texttt{]}$) represents an additional $R \circ f$-successor that may be required to satisfy the min (resp. max) constraint.

sum: It is encoded into a constraint 7.1.2. The first part of the expression $\texttt{sum}(j \in \texttt{R}[i])(\texttt{if } j \in \texttt{R}[i] \texttt{ then f}[j] \texttt{ else } 0 \texttt{ endif})$ sums the values of elements in the multiset of abstract individuals. $\texttt{sx}^{R \circ f}_{\texttt{sum}}[i]$ is the sum of additional $R \circ f$-successors to satisfy the concrete domain constraints.

min: It is encoded into a constraint 7.1.3. The first disjunct is the case that the result of the `min` function is in the multiset of abstract individuals. Then $\texttt{x}^{R \circ f}_{\texttt{min}}[i]$ is simply assigned to the minimum value of the multiset. The second disjunct is the case that the result of the **min** function is not in the multiset. Then $\texttt{x}^{R \circ f}_{\texttt{min}}[i]$ is assigned to a natural number that is less than the result of the **min** function over the actual individuals in the multiset and $\texttt{bx}^{R \circ f}_{\texttt{min}}[i]$ is a flag that is used in the calculation of count, since the count has to take into account the extra element when $\texttt{bx}^{R \circ f}_{\texttt{min}}[i]$ is true.

max: It is encoded into a constraint 7.1.4. Similar to `min`, the first disjunct is the case that the result of `max` is in the multiset. Then $\texttt{x}^{R \circ f}_{\texttt{max}}[i]$ is assigned to the maximum value of the multiset. The second disjunct is the case that the result of `max` is not in the multiset. Then $\texttt{x}^{R \circ f}_{\texttt{max}}[i]$ is assigned to a natural number that is greater than the result of `max` over the actual individuals in the multiset and $\texttt{bx}^{R \circ f}_{\texttt{max}}[i]$ is a flag that is used in the calculation of count, since count has to count the extra element when $\texttt{bx}^{R \circ f}_{\texttt{max}}[i]$ is true.

To maintain the consistency between count and sum, and max and min, constraints 7.1.5–7.1.7 need to be added.

count and sum: Constraints 7.1.5 and 7.1.6 ensure that the average value of the elements in the multiset is between min and max.

min and max: Constraint 7.1.7 ensures that the minimum value is less than or equal to the maximum value.

To check the satisfiability of concept A, a constraint $\texttt{A} \neq \emptyset$ is added (Rule **ELU-ER6**), ensuring that the set `A` is not empty. If this constraint is satisfiable w.r.t. all constraints in $\varphi^{\mathcal{T}}$, then the concept A is satisfiable.

To check subsumption of $A \sqsubseteq B$, we can add a constraint $\texttt{A} \cap (\texttt{T - B}) \neq \emptyset$. If this constraint is unsatisfiable w.r.t. all constraints in $\varphi^{\mathcal{T}}$, then $\mathcal{T} \models A \sqsubseteq B$. The following example illustrates how to encode an ontology.

Example 7.1.1. Let us use the ontology in Figures 6.4 and 6.5 for this example. Figure 7.1 shows the constraints generated for the MiniZinc encoding of the ontology from Figure 6.4 and Figure 7.2 shows some sample constraints generated for the MiniZinc encoding of the ontology from Figure 6.5.

The axioms A2–A5 are easily encoded into the constraints C1–C4 respectively. Let us consider C5, which states that for all `StateA`, for example, their feature `hours` is equal to the sum of all `hours` values of `exercise`-successors. Since there may be not enough abstract domain `exercise`-successors, we may have to account for extra successors using the slack variable `s_sum_hours`. The other states (StateB and StateC) and DailyAggregate are encoded in a similar way as StateA. The full encoding of the stream ontology in Figure 6.5 is shown in Appendix A.

Finally, in order to check whether DailyAggregate $\sqsubseteq$ GoalState, the constraint below is added to the constraint model.

```
constraint card(DailyAggregate ∩ ¬ GoalState) > 0
```

This constraint requires the `DailyAggregate ∩ ¬ GoalState` concept to be non-empty. The constraint model will therefore be unsatisfiable if and only if the concept subsumption DailyAggregate $\sqsubseteq$ GoalState holds. □

```
constraint forall(i in T)(
  i in Treadmill <-> ((hours[i] = 1) ∧ (steps[i] = 4) ∧
   (calburn[i] = 800)) );                                    (C1)
constraint forall(i in T)(
  i in FlexStrider <-> ((hours[i] = 1) ∧ (steps[i] = 3) ∧
   (calburn[i] = 700))  );                                   (C2)
constraint forall(i in T)(
  i in CrossTrainer <-> ((hours[i] = 1) ∧ (steps[i] = 3) ∧
   (calburn[i] = 750)) );                                    (C3)
constraint forall(i in T)(
  i in GoalState <->
  ((hours[i] >= 3) ∧ (hours[i] <= 5) ∧
   (steps[i] >= 10) ∧ (calburn[i] >= 2000) ∧
   (HR[i] >= 128) ∧
   (card(has[i]) + s_count_has_HR[i] +
        bool2int(b_max_has_HR[i]) +
        bool2int(b_min_has_HR[i]) = 3));                     (C4)
```

Figure 7.1: MiniZinc constraints of the ontology in Figure 6.4.

7.1.2 Finiteness of MiniZinc Models

Since CP solvers only work with finite domains, our CP-based encoding requires a bounded universe. We therefore need to calculate an upper bound on the number of individuals needed for the encoding to be complete. The following argument shows that for $\mathcal{ELU}^{(\neg)}(f,\Sigma)$, one root individual plus one individual per existential restriction are sufficient for the abstract part of the encoding.

The main proof idea is to show that, given an interpretation $\mathcal{I}$ that satisfies the abstract domain part of an $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ concept $\hat{C}$ in the context of an acyclic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox T, and which has been constructed using the completion rules from (Baader & Sattler, 2003), we can construct an interpretation $\mathcal{I}'$ that has at most $n+1$ individuals, and which also satisfies the abstract domain part of $\hat{C}$. The reason that this is possible is that any two individuals that are introduced because of the same existential restriction in the original TBox have the same concept restrictions placed on them, which means that they can be *shared.*

Example 7.1.2. Given a TBox $\mathcal{T}$ that contains an axiom $A \equiv \exists R.B$ and a concept description $\hat{C} := A \sqcap \exists R.A$, the concept $\hat{C}$ can be unfolded to $\hat{C}' := \exists R.B \sqcap \exists R.(\exists R.B)$. As can be seen, both $\exists R.B$ concepts in $\hat{C}'$ are from the same axiom in $\mathcal{T}$. Let $\mathcal{A}$ be a constraint system, a be the root individual, $\mathcal{A}(a)$ be a set of concepts that the individual a belongs to, i.e., $\mathcal{A}(a) = \{A, B, A \sqcap B, \exists R.(\hat{C} \sqcup \hat{D})...\}$. $\mathcal{A}$ is generated for $\hat{C}'$ by performing the completion rules in (Baader & Sattler, 2003). Then an individual b is introduced for B of the first conjunct $\exists R.B$ and individuals c, d are introduced for $\exists R.B$ and B in the second conjunct respectively. Considering individuals b and d, they are introduced for the existential restrictions that were unfolded from the same axiom $A \equiv \exists R.B$. Due to the absence of universal restriction, no other concept can be added to $\mathcal{A}(b) = \{B\}$ and $\mathcal{A}(d) = \{B\}$. As a consequence, the individuals b and d are indistinguishable and can be shared. In other words, the individual d can be replaced by the individual b. An upper bound on the number of individuals required for complete reasoning is therefore 3 (one root individual, one for $\exists R.A$, one for $\exists R.B$). □

```
constraint forall(i in StateA)(
 card(excercise[i] ∩ Treadmill) >= 1 ∧ (id[i] = 1) ∧ (HR[i] = 100)
 hours[i] = s_sum_hours[i] +
        sum (j in exercise[i]) (hours[i]) ∧
 calburn[i] = s_sum_calburn[i] +
        sum (j in exercise[i]) (calburn[i]) ∧
 steps[i] = s_sum_steps[i] +
        sum (j in exercise[i]) (steps[i]) ∧
 (card(has[i]) + s_count_has_hours[i] +
        bool2int(b_max_has_hours[i]) +
        bool2int(b_min_has_hours[i]) = 1) ∧
 (card(has[i]) + s_count_has_calburn[i] +
        bool2int(b_max_has_calburn[i]) +
        bool2int(b_min_has_calburn[i]) = 1) ∧
 (card(has[i]) + s_count_has_steps[i] +
        bool2int(b_max_has_steps[i]) +
        bool2int(b_min_has_steps[i]) = 1) );                    (C5)
constraint forall(i in T)(
 i in DailyAggregate <->
 (card(has[i] ∩ StateA) >= 1 ∧
  card(has[i] ∩ StateB) >= 1 ∧
  card(has[i] ∩ StateC) >= 1 ∧
  HR[i] = max (j in has[i]) (HR[j]) ∧
  hours[i] = s_sum_hours[i] +
        sum (j in exercise[i]) (hours[i]) ∧
  calburn[i] = s_sum_calburn[i] +
        sum (j in exercise[i]) (calburn[i]) ∧
  steps[i] = s_sum_steps[i] +
        sum (j in exercise[i]) (steps[i]) ∧
 (card(has[i]) + s_count_has_hours[i] +
        bool2int(b_max_has_hours[i]) +
        bool2int(b_min_has_hours[i]) = 3) ∧
 (card(has[i]) + s_count_has_calburn[i] +
        bool2int(b_max_has_calburn[i]) +
        bool2int(b_min_has_calburn[i]) = 3) ∧
 (card(has[i]) + s_count_has_steps[i] +
        bool2int(b_max_has_steps[i]) +
        bool2int(b_min_has_steps[i]) = 3) );                    (C8)
```

Figure 7.2: Sample of MiniZinc constraints of the ontology in Figure 6.5.

Theorem 7.1.1. *Given an acyclic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox $\mathcal{T}$ in normal form as described in Section 6.1.1 and an $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ concept description $\hat{C}$ such that the abstract part of $\hat{C}$ is satisfiable, then there exists an interpretation $\mathcal{I}'$ with at most $n+1$ individuals that satisfies the abstract part of $\hat{C}$, where n is the number of existential restrictions occurring in $\mathcal{T}$ and 1 is for the root individual.*

Proof. Let $\mathcal{A}$ be a constraint system, b be an individual, $P_1, ..., P_i, ...P_n$ be new fresh concept names. Since we only focus on the abstract part of the concept description $\hat{C}$, we first replace each concrete domain concept occurring in $\hat{C}$ by a distinct P_i to obtain a TBox $\mathcal{T}'$. $\mathcal{T}'$ is logically equivalent to $\mathcal{T}$, where only the abstract part is considered.

Then we assume that the concept description $\hat{C}$ is unfolded in order to get rid of $\mathcal{T}'$ and normalised into negative normal form, and the constraint system $\mathcal{A}$ is generated for $\hat{C}$ by the completion rules in (Baader & Sattler, 2003).

Now, we have an interpretation $\mathcal{I}$ that satisfies the abstract part of $\hat{C}$ from $\mathcal{A}$. Regarding the completion rules, it can be seen that individuals are *only* introduced by existential restrictions. According to the concept $\hat{C}$ unfolding, the number of existential restrictions may increase exponentially. Therefore, the interpretation $\mathcal{I}$ may have an exponential number of individuals. However, from the interpretation $\mathcal{I}$, it can be seen that any two individuals introduced for the same syntactic existential restriction from the original TBox have exactly the same constraints imposed on them, i.e., they need to satisfy exactly the same concepts. This is true since, in the absence of universal restrictions, only the concepts in the *body* of the existential restriction constrain any introduced individuals. As a result, individuals introduced because of the same existential restriction cannot be forced to be different. Therefore, these individuals can be shared. As a consequence, we can construct a smaller interpretation $\mathcal{I}'$ of $\hat{C}$, which has at most $n+1$ individuals, where n is the number of existential restrictions occurring in $\mathcal{T}$, and 1 is for the root individual. The interpretation $\mathcal{I}$ satisfies the abstract part of $\hat{C}$ if and only if $\mathcal{I}'$ does. $\square$

7.2 Correctness and Completeness of the Encoding

In this section, we show that our encoding scheme of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is sound and complete. In addition, we show that concept satisfiability and consistency checking of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is NP-COMPLETE, and limited concept subsumption checking of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is CONP-COMPLETE.

Theorem 7.2.1. *Given an acyclic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox $\mathcal{T}$ in normal form and the MiniZinc encoding, the concept A is satisfiable w.r.t. $\mathcal{T}$ if and only if the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable.*

Proof. It is a direct consequence of the following Lemmas. $\square$

Lemma 7.2.2. *(Soundness) Given an acyclic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox $\mathcal{T}$ in normal form and the MiniZinc encoding, if the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable then the concept A is satisfiable w.r.t. $\mathcal{T}$.*

The main proof idea is to construct an interpretation $\mathcal{I}$ from a solution of the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$, and then show by a straightforward inductive argument that $\mathcal{I}$ is a model for $\mathcal{T}$ and has a non-empty concept A.

The only argument in the proof that requires explanation is that for each slack variable, we can add a corresponding number of new individuals to $\mathcal{I}$ without affecting the satisfiability of any of the axioms in $\mathcal{T}$. This can be achieved easily by creating these new individuals such that they are not an instance of any concept name. Because $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ disallows negation to be on the left-hand side of axioms, such individuals trivially satisfy all axioms.

Proof. Let $\varphi^{\mathcal{T}}$ be the MiniZinc formula of $\mathcal{T}$, N_R be a set of roles in $\mathcal{T}$, and N_f be a set of features in $\mathcal{T}$. $\mathcal{S} = (\mathtt{T}^S, S_C, S_R, S_f, SX)$ is a solution tuple of the MiniZinc formula, where $\mathtt{T}^S$ represents the set of actual individuals appearing in the encoding, S_C represents a set of assignments of concept name variables $\mathtt{A}^S$, S_R represents a set of assignments of role variables $\mathtt{R}^S$, S_f represents a set of assignments of feature variables $\mathtt{f}^S$, and SX represents a set of assignments of slack variables $(\mathtt{sx}_{\Gamma}^{R\circ f})^S$, which is the value of aggregation over lacking $R \circ f$-successors. $\mathbb{N}$ represents the set of natural numbers. We must prove that there also exists a model for $\mathcal{T}$. From $\mathcal{S}$, we define an interpretation $\mathcal{I}$ as follows:

$$\Delta^{\mathcal{I}} = \{a \mid a \in \mathtt{T}^{S}\} \cup \bigcup_{R \in N_R, f \in N_f, a \in \mathtt{T}^S} \{s_i^{Rfa} | i \in 1...(\mathtt{sx}^{R\circ f}_{\mathtt{count}}\mathtt{[}a\mathtt{]})^S\}, \tag{7.2.1}$$

$$\Delta^{\mathrm{N}} = \{d \mid d \in \mathbb{N}\}, \tag{7.2.2}$$

$$A^{\mathcal{I}} = \mathtt{A}^{S}, \tag{7.2.3}$$

$$R^{\mathcal{I}} = \{(a,b) \mid a,b \in \Delta^{\mathcal{I}} \text{ and } b \in \mathtt{R[}a\mathtt{]} \cup \bigcup_{f \in N_f, a \in \mathtt{T}^S} \{(a, s_i^{Rfa}) | i \in 1...(\mathtt{sx}^{R\circ f}_{\mathtt{count}}\mathtt{[}a\mathtt{]})^S\}, \tag{7.2.4}$$

$$F^{\mathcal{I}} = f^{\mathcal{I}} \cup (\Gamma(R \circ f))^{\mathcal{I}} \tag{7.2.5}$$

$$\begin{aligned} f^{\mathcal{I}} =& \{(a,d) \mid a \in \Delta^{\mathcal{I}}, d \in \Delta^{\mathrm{N}} \text{ and } (d = \mathtt{f[}a\mathtt{]} \text{ or } d \text{ is constant natural number})\} \\ &\cup \bigcup_{R \in N_R, a \in \mathtt{T}^S} \{(s_i^{Rfa}, d_i^{Rfa}) \mid i \in 1...(\mathtt{sx}^{R\circ f}_{\mathtt{count}}\mathtt{[}a\mathtt{]})^S \text{ and } d_i^{Rfa} \in \Delta^{\mathrm{N}}\}, \end{aligned} \tag{7.2.6}$$

$$f^{\mathcal{I}}(a) = \begin{cases} d & \text{if } (a,d) \in f^{\mathcal{I}} \\ d_{Rfa_i} & \text{if } (a, d_i^{Rfa}) \in f^{\mathcal{I}} \\ \text{undefined} & \text{otherwise} \end{cases}, \tag{7.2.7}$$

$$(R \circ f)^{\mathcal{I}} = \{(a, Y_a^{R\circ f}) \mid a \in \Delta^{\mathcal{I}} \text{ and } Y_a^{R\circ f} = [\![d \mid b \in \Delta^{\mathcal{I}}, (a,b) \in R^{\mathcal{I}}, (b,d) \in f^{\mathcal{I}}]\!]\}. \tag{7.2.8}$$

$$(\Gamma(R \circ f))^{\mathcal{I}} = \{(a,e) \mid a \in \Delta^{\mathcal{I}}, e = \Gamma(Y_a^{R\circ f}) \text{ and } (a, Y_a^{R\circ f}) \in (R \circ f)^{\mathcal{I}}\}, \tag{7.2.9}$$

$$\begin{aligned} \bowtie^{\mathrm{D}} =& \{(d_1,d_2) \mid d_1, d_2 \in \Delta^{\mathrm{N}}, (a,d_1) \in F_1^{\mathcal{I}}, (a,d_2) \in F_2^{\mathcal{I}} \text{ and } d_1 \bowtie d_2 \text{ is true}\} \\ & \text{where } \bowtie \in \{\geq, <, \leq, >, =, \neq\} \end{aligned} \tag{7.2.10}$$

For non-named concepts, we define $\mathcal{I}$ such that:

$$(\sqcap_i A_i)^{\mathcal{I}} = \{a \mid a \in \Delta^{\mathcal{I}} \text{ and } a \in \bigcap_i \mathtt{A}_i\} \tag{7.2.11}$$

$$(\sqcup_j B_j)^{\mathcal{I}} = \{a \mid a \in \Delta^{\mathcal{I}} \text{ and } a \in \bigcup_j \mathtt{B}_j\} \tag{7.2.12}$$

We prove by induction over the structure of $\mathcal{T}$ that $\mathcal{I}$ is semantically consistent and it is a model of $\mathcal{T}$. For this purpose, for every axiom $\hat{C} \sqsubseteq \hat{D} \in \mathcal{T}$ in normal form and every individual a, we must prove that $\mathcal{I}$ satisfies the following condition: if $a \in \hat{C}^{\mathcal{I}}$ then $a \in \hat{D}^{\mathcal{I}}$

(i.e. $\hat{C}^{\mathcal{I}} \subseteq \hat{D}^{\mathcal{I}}$, respecting the semantics of the axiom $\hat{C} \sqsubseteq \hat{D}$) When we talk about the semantics of concepts, we always refer to Table 6.1.

The condition trivially follows from point 2, 3 and 4 of Definition 7.1.1. Let us consider the following cases of axioms: (1) $A \sqsubseteq B$, (2) $A \sqsubseteq \sqcup_j B_j$, (3) $A \sqsubseteq \exists R.B$, (4) $A \sqsubseteq \bowtie .(F_1, F_2)$, (5) $A \equiv \sqcap_j B_j$, (6) $A \equiv \sqcup_j B_j$, (7) $A \equiv \exists R.B$, and (8) $A \equiv \bowtie .(F_1, F_2)$. With respect to the axiom syntax defined in Table 6.2, where negation cannot be on the left-hand side of axioms, only B can be either B or $\neg B$. Any axiom of $\mathcal{T}$ in normal form is a sub-case of one among (1)–(8). The proof below is for (1)–(5).

(1) By hypothesis, we have $a \in A^{\mathcal{I}}$. Thus, $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (7.2.3). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ELU-ER2**) $\varphi^{\mathcal{T}}$ contains the clause ($\mathtt{A} \subseteq \mathtt{B}$) of type (7.1.8). It follows $a \in \mathtt{B}^S$ since $\varphi^{\mathcal{T}}$ is satisfiable. Therefore, $a \in B^{\mathcal{I}}$ by (7.2.3).

(2) By hypothesis, we have $a \in A^{\mathcal{I}}$. Then $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (7.2.3). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ELU-ER2**) $\varphi^{\mathcal{T}}$ contains the clause ($\mathtt{A} \subseteq (\bigcup_j \mathtt{B}_j)$) of type (7.1.8). It follows that $a \in \bigcup_j \mathtt{B}_j$ because $\varphi^{\mathcal{T}}$ is satisfiable. Thus, $a \in (\sqcup_j B_j)^{\mathcal{I}}$ by definition of $\mathcal{I}$ (7.2.12).

(3) By hypothesis, we have $a \in A^{\mathcal{I}}$. Therefore, $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (7.2.3). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ELU-ER3**) $\varphi^{\mathcal{T}}$ contains the constraint $\mathtt{A} \subseteq \{a \mid \forall a \in \mathtt{T} \wedge \mathtt{card(R[}a\mathtt{]} \cap \mathtt{B)} \geq 1\}$ of type (7.1.9). Due to this constraint, $\varphi^{\mathcal{T}}$ can be satisfiable only if $a \in \{a \mid \forall a \in \mathtt{T} \wedge \mathtt{card(R[}a\mathtt{]} \cap \mathtt{B)} \geq 1\}$ is true. Thus, there exists at least one individual b such that $b \in \mathtt{T}^S$, $b \in \mathtt{R}^S\mathtt{[}a\mathtt{]}$ and $b \in \mathtt{B}^S$ because $\varphi^{\mathcal{T}}$ is satisfiable. By definitions of $\mathcal{I}$ (7.2.4) and (7.2.3), $(a, b) \in R^{\mathcal{I}}$ and $b \in B^{\mathcal{I}}$. As a result, $a \in (\exists r.B)^{\mathcal{I}}$.

(4) By hypothesis, we have $a \in A^{\mathcal{I}}$. Therefore, $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (7.2.3). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ELU-ER5**) $\varphi^{\mathcal{T}}$ contains the constraint $\mathtt{A} \subseteq \{a \mid \forall a \in \mathtt{T} \wedge \mathsf{tran}(F_1) \bowtie \mathsf{tran}(F_2)\}$ of type (7.1.11). Due to this constraint, $\varphi^{\mathcal{T}}$ can be satisfiable only if $a \in \{a \mid \forall a \in \mathtt{T} \wedge \mathsf{tran}(F_1) \bowtie \mathsf{tran}(F_2)\}$ is true. Because $\varphi^{\mathcal{T}}$ is satisfiable, let us consider the following cases of features: (i) $F = f$, (ii) $F = d$, and (iii) $F = \Gamma(R \circ f)$. There is a tran function for each case (**ELU-ER5**). Due to restriction 3, the concrete domain is always finite. Basically, we have d_1 as a value for F_1 and d_2 as a value for F_2. For simplification, we will use d instead of d_1 and d_2 in this proof.

Let $\Gamma(Y_a^{R \circ f})$ be the result of aggregation over a multiset $Y_a^{R \circ f}$, where $Y_a^{R \circ f} = [\![e_i | \forall b_i \in \Delta^{\mathcal{I}}, (a, b_i) \in R^{\mathcal{I}}, (b_i, e_i) \in f^{\mathcal{I}}]\!]$, and $(a, Y_a^{R \circ f}) \in (R \circ f)^{\mathcal{I}}$. For simplification, we use $\Gamma(Y_a^{R \circ f})$ to represent the constraint that is used to calculate the result of aggregations. For example, $\Gamma(Y_a^{R \circ f})$ represents $\mathtt{sum(}j \in \mathtt{R[}i\mathtt{])(if}\ j \in \mathtt{R[}i\mathtt{]}\ \mathtt{then\ f[}j\mathtt{]\ else}\ 0\ \mathtt{endif)}$, where Γ is sum. Now we consider all possible aggregations.

(i) Since $a \in \mathtt{A}^S$, $\mathsf{tran}(F) = \mathtt{f[}a\mathtt{]}$, and $\{\mathtt{f[}a\mathtt{]}\} \subseteq \mathbb{N}$, there exists one concrete domain individual d such that $d = \mathtt{f[}a\mathtt{]}$ and $d \in \mathbb{N}$. By definition of $\mathcal{I}$ (7.2.6) and (7.2.5), $(a, d) \in f^{\mathcal{I}}$ and $(a, d) \in F^{\mathcal{I}}$

(ii) Since $a \in \mathtt{A}^S$, $\mathsf{tran}(F) = \mathtt{d}$, and $\{\mathtt{d}\} \subseteq \mathbb{N}$, there exists one concrete domain individual d such that $d = \mathtt{d}$ and $d \in \mathbb{N}$. By definition of $\mathcal{I}$ (7.2.6) and (7.2.5), $(a, d) \in f^{\mathcal{I}}$ and $(a, d) \in F^{\mathcal{I}}$.

(iii) Since $a \in \mathtt{A}^S$, $\mathsf{tran}(F) = \mathtt{x}_{\Gamma}^{R \circ f}\mathtt{[}a\mathtt{]}$, there exists one concrete domain individual d such that $d = \mathtt{x}_{\Gamma}^{R \circ f}\mathtt{[}a\mathtt{]}$ and $d \in \mathbb{N}$. However, an abstract individual a may not have enough R-successors that have an f-successor as required by the solution for $\Gamma(Y_a^{R \circ f})$. Due to the absence of universal qualification, we can add R-successors s_{Rfa_i} for a and $f^{\mathcal{I}}(s_{Rfa_i})$ to

$Y_a^{R\circ f}$. By definition of $\mathcal{I}$ (7.2.7), (7.2.9) and (7.2.5), $(a,d) \in (\Gamma(R \circ f))^{\mathcal{I}}$ and $(a,d) \in F^{\mathcal{I}}$. Note that, this proof is correct if the value of all aggregations is consistent.

min : Since Γ is min, (**ELU-ER1**) $\varphi^{\mathcal{T}}$ contains a constraint of type (7.1.3). $\varphi^{\mathcal{T}}$ can be satisfiable only if either $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f})$ is true or $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} < \Gamma(Y_a^{R\circ f}) \wedge \mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is true. If $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} < \Gamma(Y_a^{R\circ f}) \wedge \mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is true, $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} < \Gamma(Y_a^{R\circ f})$ is true, $\mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is true and the minimum value is not in the multiset $Y_a^{R\circ f}$. Then $\mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ will be used to calculate count function.

max : Since Γ is max, (**ELU-ER1**) $\varphi^{\mathcal{T}}$ contains a constraint of type (7.1.4). $\varphi^{\mathcal{T}}$ can be satisfiable only if either $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f})$ is true or $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} > \Gamma(Y_a^{R\circ f}) \wedge \mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is true. If $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} > \Gamma(Y_a^{R\circ f}) \wedge \mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is true, $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} > \Gamma(Y_a^{R\circ f})$ is true, $\mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is true and the maximum value is not in the multiset $Y_a^{R\circ f}$. Then $\mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ will be used to calculate count function.

min and max : Since the value of min is always less than that of max, (**ELU-ER1**) $\varphi^{\mathcal{T}}$ contains a clause of type (7.1.7).

count : Since Γ is count, (**ELU-ER1**) $\varphi^{\mathcal{T}}$ contains a clause of type (7.1.1). $\varphi^{\mathcal{T}}$ can be satisfiable only if $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) + \mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} + \mathtt{bool2int}(\mathtt{bx}_{\mathtt{max}}^{R\circ f}\mathtt{[}a\mathtt{]}) + \mathtt{bool2int}(\mathtt{bx}_{\mathtt{min}}^{R\circ f}\mathtt{[}a\mathtt{]})$ is true. As mentioned in the previous cases, $\mathtt{bx}_{\mathtt{max}}^{R\circ f}\mathtt{[}a\mathtt{]}$ and $\mathtt{bx}_{\mathtt{min}}^{R\circ f}\mathtt{[}a\mathtt{]}$ are used to count whether the minimum and/or maximum are present in the multiset, and therefore need to be taken into account to calculate the count function. Since $\mathtt{bx}_{\mathtt{max}}^{R\circ f}\mathtt{[}a\mathtt{]}$ and $\mathtt{bx}_{\mathtt{min}}^{R\circ f}\mathtt{[}a\mathtt{]}$ are Boolean variables, they can be converted to natural numbers using the **bool2int** function. Therefore, if either $\mathtt{bx}_{\mathtt{max}}^{R\circ f}\mathtt{[}a\mathtt{]}$ or $\mathtt{bx}_{\mathtt{min}}^{R\circ f}\mathtt{[}a\mathtt{]}$ is true, the value of $\mathtt{bool2int}(\mathtt{bx}_{\mathtt{max}}^{R\circ f}\mathtt{[}a\mathtt{]})$ or $\mathtt{bool2int}(\mathtt{bx}_{\mathtt{min}}^{R\circ f}\mathtt{[}a\mathtt{]})$ will be 1, otherwise it will be 0. This will happen when the minimum or maximum value is not in the multiset Y. $\mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is a so-called *slack variable*, which holds the number of individuals added to $\mathtt{T}^S$ to satisfy $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) + \mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} + \mathtt{bool2int}(\mathtt{bx}_{\mathtt{max}}^{R\circ f}\mathtt{[}a\mathtt{]}) + \mathtt{bool2int}(\mathtt{bx}_{\mathtt{min}}^{R\circ f}\mathtt{[}a\mathtt{]})$ when an individual a has less R-successors than required.

sum : Since Γ is sum, (**ELU-ER1**) $\varphi^{\mathcal{T}}$ contains a clause of type (7.1.2). $\varphi^{\mathcal{T}}$ can be satisfiable only if $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) + \mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is true. $\mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is a *slack variable*, which is a natural number used to satisfy $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) + \mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ when an individual a has less R-successors than required.

count and sum : Since the values of count and sum need to be consistent, (**ELU-ER1**) $\varphi^{\mathcal{T}}$ contains clauses of type (7.1.5) and (7.1.6) to ensure the average of each element in Y is between the max value and min value.

By definition of $\mathcal{I}$ (7.2.10), $(d_1, d_2) \in \bowtie^{\mathrm{D}}$. Thus, $a \in (\bowtie .(F_1, F_2))^{\mathcal{I}}$

(5) By hypothesis, we have $a \in A^{\mathcal{I}}$. Then $a \in \mathtt{A}^S$ by definition of $\mathcal{I}$ (7.2.3). Since $\mathtt{A}$ is in $\varphi^{\mathcal{T}}$, (**ELU-ER4**) $\varphi^{\mathcal{T}}$ contains a constraint ($\mathtt{A} = (\bigcap_j \mathtt{B}_j)$) of type (7.1.10). It follows that $a \in \bigcap_j \mathtt{B}_j$ because $\varphi^{\mathcal{T}}$ is satisfiable. Thus, $a \in (\sqcap_j B_j)^{\mathcal{I}}$ by definition of $\mathcal{I}$ (7.2.11).

For $\neg B$, by hypothesis, let $a \in (\neg B)^{\mathcal{I}}$. By definition of (7.2.3), $a \in \Delta^{\mathcal{I}}$ and $a \in \mathtt{T}^S - \mathtt{B}^S$. It follows $a \notin B^{\mathcal{I}}$. Thus, $a \in \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$. For axioms (6)–(8), the lemma is easy to prove by using the same approach as (2)–(4). The only difference is to change subset $\subseteq$ to equivalence $=$.

Finally, we prove that if $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable, then there exists an interpretation $\mathcal{I}$, such that $\mathcal{I}$ is a model for $\mathcal{T}$ and $A^{\mathcal{I}} \neq \emptyset$. $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable if and only if $\varphi^{\mathcal{T}}$ is satisfiable. Since $\varphi^{\mathcal{T}}$ is satisfiable, there exists an individual a such that $a \in \mathtt{A}^{\mathcal{S}}$. By definition (7.2.3), we have $a \in A^{\mathcal{I}}$. Thus, the concept A is satisfiable w.r.t. $\mathcal{T}$. □

Lemma 7.2.3. *(Completeness) Given an acyclic $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBox $\mathcal{T}$ in normal form and the MiniZinc encoding, if the concept A is satisfiable w.r.t. $\mathcal{T}$ then the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable.*

The main idea of this proof is to construct a solution $\mathcal{S}$ from a model $\mathcal{I}$ of $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$, and then show that $\mathcal{S}$ is a solution of $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$. We assume that an interpretation $\mathcal{I}'$ is constructed by the completion rules in (Baader & Sattler, 2003) to satisfy the abstract part of $\mathcal{T}$. However, certain individuals in $\mathcal{I}'$ may not have enough R-successors that have a f-successor to satisfy the aggregations. Thus, the *lacking* $R \circ f$-successors of individual a ($a \xrightarrow{R} b \xrightarrow{f} d$), which are composite successors of R and f, need to be added to satisfy the concrete domain part of $\mathcal{T}$. Finally, a model $\mathcal{I}$ is obtained by adding additional individuals to satisfy the concrete domain part to $\mathcal{I}'$. Without loss of generality we can assume that $\mathcal{I}$ is chosen such that these lacking successors do not affect the abstract domain, because of the absence of universal quantification and the axiom syntax defined in Table 6.2, where negation cannot be on the left-hand side of axioms.

In the MiniZinc model, the result of each aggregation Γ of the lacking $R \circ f$-successors of individual a is represented by a *slack variable* $\mathtt{sx}_{\Gamma}^{R\circ f}\mathtt{[}a\mathtt{]}$. It represents the number of fresh lacking $R \circ f$-successors that need to be added to satisfy $\varphi^{\mathcal{T}}$ when Γ is count. When Γ is sum, $\mathtt{sx}_{\Gamma}^{R\circ f}\mathtt{[}a\mathtt{]}$ is the sum of fresh lacking $R \circ f$-successors that need to be added. Additionally, for each individual a, $\mathtt{bx}_{\Gamma}^{R\circ f}\mathtt{[}a\mathtt{]}$ is a Boolean variable that represents the absence of the min (or max) value over the multiset of $R \circ f$ successors of a, giving rise to the need to add one lacking $R \circ f$-successor.

Proof. Given that the concept A is satisfiable w.r.t. $\mathcal{T}$, there exists a model $\mathcal{I}$ for $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$. For each individual $a \in \Delta^{\mathcal{I}}$, let $Y_a^{R\circ f}$ be the multiset of $R \circ f$-successors generated by the completion rules, $X_a^{R\circ f} \supseteq Y_a^{R\circ f}$ be the multiset of *all* $R \circ f$-successors, including the additional ones required to satisfy the concrete domain axioms, $d_i^{Rfa} \in X_a^{R\circ f} \setminus Y_a^{R\circ f}$ be the ith lacking $R \circ f$-successor, and b_i^{Rfa} be the ith R-successor connecting a and d_i^{Rfa} (i.e., $a \xrightarrow{R} b_i^{Rfa} \xrightarrow{f} d_i^{Rfa}$). **bool2int** is a function mapping Boolean values to $\{0,1\}$, i.e., $\mathbf{bool2int}(\mathsf{true}) = 1$, and $\mathbf{bool2int}(\mathsf{false}) = 0$. The completion rules in (Baader & Sattler, 2003) are sound, complete and terminating. Therefore, we can construct a model $\mathcal{I}$ for $\mathcal{T}$, and from $\mathcal{I}$ build a solution $\mathcal{S}$ for $\varphi^{\mathcal{T}}$.

$\mathcal{S} = (\mathtt{T}^{\mathcal{S}}, S_C, S_R, S_f, SX)$ is a solution tuple of a MiniZinc formula as introduced above. From $\mathcal{I}$, we define the solution $\mathcal{S}$ as follows:

$$\texttt{T}^S = \Delta^{\mathcal{I}} \setminus \bigcup_{(a,X_a^{R\circ f})\in(R\circ f)^{\mathcal{I}}} \{b_i^{Rfa} | d_i^{Rfa} \in X_a^{R\circ f} \setminus Y_a^{R\circ f}\} \tag{7.2.13}$$

$$\texttt{A}^S = A^{\mathcal{I}} \tag{7.2.14}$$

$$\texttt{R}^S\texttt{[}a\texttt{]} = \{b \mid a,b \in \texttt{T}^S \text{ and } (a,b) \in R^{\mathcal{I}}\} \tag{7.2.15}$$

$$\texttt{f}^S\texttt{[}a\texttt{]} = \begin{cases} d & \text{if } (a,d) \in f^{\mathcal{I}} \\ d_c & \text{if } (a,d_c) \in f^{\mathcal{I}} \wedge d_c \in \mathbb{N} \\ \text{undefined} & \text{otherwise} \end{cases} \tag{7.2.16}$$

$$(\texttt{sx}_\Gamma^{R\circ f}\texttt{[}a\texttt{]})^S = \Gamma(X_a^{R\circ f} \setminus Y_a^{R\circ f}) \tag{7.2.17}$$

Now, we must prove that $\mathcal{S}$ satisfies all the constraints of $\varphi^{\mathcal{T}}$ such that $a \in \Delta^{\mathcal{I}}$ for every type of constraint from (7.1.1) to (7.1.11).

Constraints of type (7.1.8) represent the encoding of axioms $A \sqsubseteq \sqcup_m B_m$. Let us consider the binary case $A \sqsubseteq B_1 \sqcup B_2$, where A, B_1 and B_2 are concept names (the proof is easily generalised). The axiom is encoded to $\texttt{A} \subseteq \texttt{B}_1 \cup \texttt{B}_2$. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $A^{\mathcal{I}} \subseteq (B_1 \sqcup B_2)^{\mathcal{I}}$. Thus, if $a \in A^{\mathcal{I}}$, then $a \in B_1^{\mathcal{I}} \cup B_2^{\mathcal{I}}$. It follows by 7.2.14 that $a \in \texttt{A}^S$ and either $a \in \texttt{B}_1^S$ or $a \in \texttt{B}_2^S$. Hence, the constraint is satisfied.

Constraints of type (7.1.9) represent the encoding of axioms $A \sqsubseteq \exists R.B$. We must show that the clause $\texttt{A} \subseteq \{a \mid \forall a \in \texttt{T} \wedge \texttt{card(R[}a\texttt{]} \cap \texttt{B)} \geq 1\}$ of type (7.1.9) is satisfied. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $A^{\mathcal{I}} \subseteq (\exists R.B)^{\mathcal{I}}$. Thus, if $a \in A^{\mathcal{I}}$, then $a \in (\exists R.B)^{\mathcal{I}}$. Since $a \in (\exists R.B)^{\mathcal{I}}$, $(a,b) \in R^{\mathcal{I}}$ and $b \in B^{\mathcal{I}}$ such that $b \in \Delta^{\mathcal{I}}$. By definition of $\mathcal{S}$ (7.2.14) and (7.2.15), we have $a \in \texttt{A}^S$, $b \in \texttt{R}^S\texttt{[}a\texttt{]}$, and $b \in \texttt{B}^S$. As a result, the constraint is satisfied.

Constraints of type (7.1.10) represent the encoding of axioms $A \equiv \sqcap_m B_m$). Again, consider the binary case $A \equiv B_1 \sqcap B_2$, where A, B_1 and B_2 are basic concepts, which is encoded to $\texttt{A} = \texttt{B}_1 \cap \texttt{B}_2$. Since $a \in \Delta^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$, it holds $A^{\mathcal{I}} = (B_1 \sqcap B_2)^{\mathcal{I}}$. Thus, if $a \in A^{\mathcal{I}}$, then $a \in B_1^{\mathcal{I}} \cap B_2^{\mathcal{I}}$. This follows by 7.2.14 that $a \in \texttt{A}^S$ and $a \in \texttt{B}_1^S$ and $a \in \texttt{B}_2^S$. Hence, the constraint is satisfied.

For the concrete domain part, the constraint of type (7.1.11) $\texttt{A} \subseteq \{a \mid \forall a \in \texttt{T} \wedge \mathsf{tran}(F_1) \bowtie \mathsf{tran}(F_2)\}$ represents the encoding of an axiom $A \sqsubseteq \bowtie .(F_1,F_2)$. Since $\mathcal{I}$ is a model of $\mathcal{T}$, it holds that $A^{\mathcal{I}} \subseteq (\bowtie .(F_1,F_2))^{\mathcal{I}}$. Thus, if $a \in A^{\mathcal{I}}$, then $a \in (\bowtie .(F_1,F_2))^{\mathcal{I}}$. Let us consider the following cases of features F (F_1 or F_2): (i) $F = f$, (ii) $F = d$, and (iii) $F = \Gamma(R \circ f)$. tran is an encoding function for each case. Due to restriction 3 in Section 6.1, the concrete domain is always finite. Basically, we have d_1 as a value for F_1 and d_2 as a value for F_2. For simplicity, we will use c instead of d_1 and d_2 in this proof.

(i) Since $a \in A^{\mathcal{I}}$, $\mathsf{tran}(F) = \texttt{f[}a\texttt{]}$, there exists one concrete domain individual c such that $(a,c) : f$ and $c \in \mathbb{N}$. By definition (7.2.16), $\texttt{f}^S\texttt{[}a\texttt{]} = c$.

(ii) Since $a \in A^{\mathcal{I}}$, $\mathsf{tran}(F) = \mathsf{d}$, there exists one concrete domain individual c such that $(a,c) : f$ and $c \in \mathbb{N}$. By definition (7.2.16), $\mathsf{d}^S = c$.

(iii) Since $a \in A^{\mathcal{I}}$, $\mathsf{tran}(F) = \texttt{x}_\Gamma^{R\circ f}\texttt{[}a\texttt{]}$, there exists one concrete domain individual c such that c is the value of aggregating over $X_a^{R\circ f}$ for Γ. In the case that an abstract individual a does not have enough $R \circ f$-successors required by the solution for $X_a^{R\circ f}$, new $R\circ f$-successors need to be added and the aggregation result over these new $R\circ f$-successors will be recorded in a slack variable $\mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$.

Let $\Gamma(Y_a^{R\circ f})$ be the result of aggregation over $Y_a^{R\circ f}$, where $Y_a^{R\circ f} = [\![e_i | \forall b_i \in \Delta^{\mathcal{I}}, (a, b_i) \in R^{\mathcal{I}}, (b_i, e_i) \in f^{\mathcal{I}}]\!]$, and $(a, Y_a^{R\circ f}) \in (R\circ f)^{\mathcal{I}}$. For simplification, we use $\Gamma(Y_a^{R\circ f})$ to represent the constraint that is used to calculate the result of aggregations. For example, $\Gamma(Y_a^{R\circ f})$ represents $\mathtt{sum}(j \in \mathtt{R[}i\mathtt{]})(\mathbf{if}\ j \in \mathtt{R[}i\mathtt{]}\ \mathbf{then}\ \mathtt{f[}j\mathtt{]}\ \mathbf{else}\ 0\ \mathbf{endif})$, where Γ is sum. Now we consider all possible aggregations.

min: Since Γ is min, we must prove that the constraint $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) \vee (\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} < \Gamma(Y_a^{R\circ f}) \wedge \mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]})$ is satisfied. The first disjunct is for the case where no additional $R \circ f$-successors are required: since $(a, X_a^{R\circ f}) \in (R \circ f)^{\mathcal{I}}$, $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f})$ is true. The second disjunct is necessary for the case where a lacking $R \circ f$-successor is added because the minimum value is not in $Y_a^{R\circ f}$, i.e., $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} < \Gamma(Y_a^{R\circ f})$ is true. The Boolean variable $\mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$) signals that the minimum value is not in the multiset $Y_a^{R\circ f}$.

max: Since Γ is max, we must prove that the constraint $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) \vee (\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} > \Gamma(Y_a^{R\circ f}) \wedge \mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]})$ is satisfied. The first disjunct is for the case where no additional $R \circ f$-successors are required: since $(a, X_a^{R\circ f}) \in (R \circ f)^{\mathcal{I}}$, $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f})$ is true. The second disjunct is necessary for the case where a lacking $R \circ f$-successor is added because the maximum value is not in $Y_a^{R\circ f}$, i.e., $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} > \Gamma(Y_a^{R\circ f})$ is true. The Boolean variable $\mathtt{bx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$) signals that the minimum value is not in the multiset $Y_a^{R\circ f}$.

min and max: Since the value of min is always no greater than max in $\mathcal{I}$, the constraint $\mathtt{min}(X_a^{R\circ f}) \leq \mathtt{max}(X_a^{R\circ f})$ is trivially satisfied.

count: Since Γ is count, we must prove that the clause $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) + \mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} + \mathtt{bool2int}(\mathtt{bx}_{\mathtt{max}}^{R\circ f}\mathtt{[}a\mathtt{]}) + \mathtt{bool2int}(\mathtt{bx}_{\mathtt{min}}^{R\circ f}\mathtt{[}a\mathtt{]})$ is satisfied. $\Gamma(Y_a^{R\circ f})$ represents the number of actual $R \circ f$-successors in $\mathcal{I}$, $\mathtt{bool2int}(\mathtt{bx}_{\mathtt{min}}^{R\circ f}[a])$ (resp. $\mathtt{bool2int}(\mathtt{bx}_{\mathtt{max}}^{R\circ f}[a])$) represents the one additional lacking $R \circ f$-successor that may be required to satisfy the min (resp. max) constraint. $\mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is the slack variable that represents the number of additional lacking $R\circ f$-successors. Thus, since $(a, X_a^{R\circ f}) \in (R\circ f)^{\mathcal{I}}$, $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) + \mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} + \mathtt{bool2int}(\mathtt{bx}_{\mathtt{max}}^{R\circ f}\mathtt{[}a\mathtt{]}) + \mathtt{bool2int}(\mathtt{bx}_{\mathtt{min}}^{R\circ f}\mathtt{[}a\mathtt{]})$ is true.

sum: Since Γ is sum, we must prove that the clause $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) + \mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is satisfied. The slack variable $\mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ records the sum of the lacking $R\circ f$-successor values that may be required to satisfy the solution for $X_a^{R\circ f}$. Thus, since $(a, X_a^{R\circ f}) \in (R \circ f)^{\mathcal{I}}$, $\mathtt{x}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]} = \Gamma(Y_a^{R\circ f}) + \mathtt{sx}_\Gamma^{R\circ f}\mathtt{[}a\mathtt{]}$ is true.

count and sum: To maintain the D-consistency of the concrete domain (Lemma 6.4.4), especially between count and sum, the average value of elements in $X_a^{R\circ f}$ needs to be between min and max.

Then we obtain the values of $\mathsf{tran}(F_1)$ and $\mathsf{tran}(F_2)$ from $\mathcal{I}$. Given that $A^{\mathcal{I}} \subseteq (\bowtie .(F_1, F_2))^{\mathcal{I}}$ is true in $\mathcal{I}$, the constraint $\mathtt{A} \subseteq \{a \mid \forall a \in \mathtt{T} \wedge \mathsf{tran}(F_1) \bowtie \mathsf{tran}(F_2)\}$ is satisfied.

Regarding the axiom syntax defined in Table 6.2, where negation cannot be on the left-hand side of axioms, $\neg A$ can only occur on the right-hand of axioms. Let $a \in (\neg A)^{\mathcal{I}}$.

By definition of (7.2.14), $a \in \mathtt{T}^S$ and $a \in \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$. It follows $a \notin A^{\mathcal{I}}$. Thus, $a \in \mathtt{T}^S - \mathtt{A}^S$. For the equivalence ($\equiv$), the lemma is easy to prove by using the same approach as $\sqsubseteq$.

Finally, since there exists a model $\mathcal{I}$ for $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$, there exists an individual $a \in \Delta^{\mathcal{I}}$ such that $a \in A^{\mathcal{I}}$. By definition 7.2.14, we have $a \in \mathtt{A}^S$. Hence, the MiniZinc formula $\varphi^{\mathcal{T}} \wedge (\mathtt{A} \neq \emptyset)$ is satisfiable.

□

With the above syntactic restrictions in Section 2.2, we can show that *concept satisfiability* for $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is non-trivial. The proof is through a straightforward reduction from 3-SAT, which is known to be NP-HARD, as shown in Theorem 7.2.4.

Theorem 7.2.4. *The concept satisfiability of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is* NP-HARD.

Proof. For a 3-SAT formula in CNF $C = c_1 \wedge c_2 \wedge \ldots \wedge c_n$, where c_i is a disjunction of 3 literals, drawn from $x_1, x_2, ..., x_m$ and their negations, $\neg x_1, \neg x_2, ..., \neg x_m$. We can encode this formula into an $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ concept L as follows.

For each variable x_i in C, we create a corresponding named concept c_{x_i}. Each positive (resp. negative) literal is encoded as the corresponding positive (resp. negated) named concept. Each clause c_i is encoded as a disjunctive concept expression of the three (positive or negated) concepts corresponding to the literals in that clause. Next, the formula C is encoded as a conjunction of the above disjunctive concept expressions, and this conjunction is our concept L. Hence, checking the satisfiability of C is reduced to checking the satisfiability of L. □

From Theorem 7.2.1 and 7.2.4, we can show that concept satisfiability and consistency checking are NP-COMPLETE, and limited concept subsumption is coNP-COMPLETE.

Corollary 7.2.5. *The concept satisfiability problem of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is* NP-COMPLETE.

Proof. Since we show that the algorithm for encoding the concept satisfiability of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ into MiniZinc is sound, complete and polynomial in size and time, and the constraint satisfaction problem (i.e., MiniZinc) is NP-COMPLETE (Feder & Hell, 2006), the concept satisfiability problem of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is in fact NP-COMPLETE. □

Corollary 7.2.6. *The consistency checking problem of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is* NP-COMPLETE.

Proof. The consistency checking problem can be reduced to the concept satisfiability problem in polynomial size and time. Since concept satisfiability of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is NP-COMPLETE, consistency checking of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is also NP-COMPLETE. □

Corollary 7.2.7. *The limited concept subsumption checking problem of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is* coNP-COMPLETE.

Proof. The limited concept subsumption checking problem can be reduced to the concept satisfiability problem in polynomial size and time. Since concept satisfiability of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is NP-COMPLETE, limited concept subsumption checking of $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ is also NP-COMPLETE. □

7.3 Optimisations

The encodings presented above translate each syntactic construct of a DL into a corresponding MiniZinc construct. In conjunction with a suitable encoding of the reasoning task, we can therefore use the readily available MiniZinc solvers as ontology reasoners.

One of the drawbacks of the encoding $\mathcal{ELU}^{(\neg)}(f,\Sigma)$2MiniZinc is that the MiniZinc model contains *symmetries*, which may require CP solvers to explore essentially identical subtrees multiple times, leading to a large increase in runtime required for solving the model. In addition, the choice of variable and value ordering has been shown to have a large effect on the size of the search tree, and thus to be important for effectively solving many CSP problems (Ginsberg et al., 1990; Bacchus & Van Run, 1995; Gent et al., 1996). In this section, we propose techniques to address these two issues in order to reduce the size of the search space for encoded DL problems.

7.3.1 The Need for Symmetry Breaking and Search Heuristics

For a CP expert, it will be quite obvious that the encoding in Definition 7.1.1 introduces symmetries by identifying abstract individuals with natural numbers. In any solution to the MiniZinc model, we can swap the names (i.e., the natural numbers identifying the individuals) of any pair of individuals to produce again a solution, as long as we take care and also swap all of their feature values.

The encoding can therefore be extended with *symmetry breaking constraints*, avoiding the unnecessary exploration of symmetric states by the constraint solvers. The general idea of symmetry breaking was described in Section 4.3.2. In our case, the concept names are encoded to set-valued variables. In general, breaking value symmetries like this one can be achieved by imposing a lexicographic order

```
[i in A, i in B, i in C, ...] >= [i+1 in A, i+1 in B, i+1 in C, ...]
```

where `i` ranges over the possible set elements and `A,B,C,...` are the set-valued variables.

The choice of search heuristics for the encoding $\mathcal{ELU}^{(\neg)}(f,\Sigma)$2MiniZinc may be equally important. The general idea of search heuristics was described in Section 4.3.3. If the CP solver selects the wrong concepts to start search with, it may explore the search space that does not lead to any solution. In order to help the CP solvers, in general, the *search heuristic* is imposed to make a better choice of concepts.

7.3.2 Symmetry Breaking Constraints

In order to break some symmetries in the MiniZinc model from the encoding $\mathcal{ELU}^{(\neg)}(f,\Sigma)$2MiniZinc, we propose static symmetry breaking constraints as follows. Assume that a TBox $\mathcal{T}$ is encoded to a MiniZinc model `M` and `M` is satisfiable.

SB1 Testing satisfiability of a concept: In order to check the satisfiability of a concept A in $\mathcal{T}$, the constraint **`card(A) > 0`** is added to `M`, resulting in a model `M′`. This constraint means that the cardinality of A must be greater than 0. The concept A is satisfiable if and only if there is a solution for `M′`, which is satisfiable only if **`card(A) > 0`** is satisfiable. Regarding this constraint, if there are n individuals in `M′`, the CP solver may need to try adding each of these n individuals, or any combination of these, to A. Instead of the constraint **`card(A) > 0`**, we can force individual `1` to be in `A` (i.e., **`1 in A`**) without loss of generality, because all individuals are symmetric. In this case, the CP solver does not need to try n individuals, it needs to try only the individual `1`. In addition, we can enforce that **`forall (i in 2..n) (i in A -> i-1 in A)`**, since all individuals except `1` are also symmetric.

SB2 Testing concept subsumption: If we want to check concept subsumption of $A \sqsubseteq B$, the constraint **`card(A intersect (T diff B)) > 0`** is added to `M`, resulting in `M′`. We reduce concept subsumption to concept unsatisfiability checking, i.e., checking $A \sqsubseteq B$ is reduced to checking unsatisfiability of $A \sqcap \neg B$. $\mathcal{T} \models A \sqsubseteq B$ if and only if there is no solution for `M′`. `M′` has no solution only if the constraint **`card(A`**

`intersect (T diff B)) > 0` is unsatisfiable. Since we prove subsumption by testing unsatisfiability of `card(A intersect (T diff B)) > 0`, we can use the same symmetry breaking constraints as above since `A` cannot be empty. In addition, we know that individual `1` cannot be in `B`, so we can add that constraint. Furthermore, we can add the constraint `2 in B \/ card(B)=0`, since all other individuals are still symmetric.

SB3 Lexicographical Ordering on Individuals: We can view the assignments of individuals to concepts as a matrix of Boolean values (i.e., 1 and 0), where the columns of the matrix are the individual names (`1`, `2`, `3`, ...) and the rows of the matrix are set variables corresponding to the concept names (`A`, `B`, `C`, ...) as shown in Figure 7.3. The value 1 in the matrix represents the presence of an individual in the corresponding set variable. As mentioned, all individuals are symmetric. The symmetry can be broken by ordering the columns of the matrix lexicographically from left to right. The intuition is that the small individuals are always assigned to the set variables before the large individuals, i.e., the individual `1` is always assigned to a concept before the individual `2`.

Now, we treat columns as vectors. Let $\mathtt{v_i}$ be the vector of `0`s and `1`s of column `i`. For example, in Figure 7.3a, the value of $\mathtt{v_1}$ is $[\mathtt{1,0,0}]$. Then we can post the lexicographical ordering constraint $\mathtt{v_i} >=_{\mathtt{lex}} \mathtt{v_{i+1}} >=_{\mathtt{lex}} \mathtt{v_{i+2}} >=_{\mathtt{lex}} \ldots >=_{\mathtt{lex}} \mathtt{v_n}$, where `n` is the number of columns (the number of individuals), to ensure that the small individuals are always assigned to the set variables first. Let us describe this through an example.

Example 7.3.1. Given a TBox $\mathcal{T}$ that contains the axioms $A \sqsubseteq \exists R.B$ and $B \sqsubseteq \exists R.C$, let us check whether the concept A is satisfiable. We encode $\mathcal{T}$ using the encoding $\mathcal{ELU}^{(\neg)}(f, \Sigma)$2MiniZinc to a MiniZinc model `M`. This example has 3 concepts A, B, C, which are encoded to set variables `A`, `B`, `C`, and 3 individuals $1, 2, 3$, which are encoded to natural numbers `1`, `2`, `3`. Figure 7.3 shows the matrices of three solutions of this example. In Figure 7.3a, the individuals `1`, `2`, and `3` are assigned to the set variables `A`, `B`, and `C` respectively. If we now flip the solution 1 horizontally, we will get another solution 2 of `M`, where the individuals `3`, `2`, and `1` are assigned to the set variables `A`, `B`, and `C` respectively. Solution 2 is a symmetric version of Solution 1.

To eliminate the symmetric solutions, as mentioned above, the columns are ordered lexicographically by posting a lexicographical ordering constraint $\mathtt{v_1} >=_{\mathtt{lex}} \mathtt{v_2} >=_{\mathtt{lex}} \mathtt{v_3}$. Considering Figure 7.3a, $\mathtt{v_1} = [\mathtt{1,0,0}]$, $\mathtt{v_2} = [\mathtt{0,1,0}]$, and $\mathtt{v_3} = [\mathtt{0,0,1}]$. As can be seen, with the lexicographical ordering constraint, only Solution 1 remains a solution of `M` since it satisfies the constraint $\mathtt{v_1} >=_{\mathtt{lex}} \mathtt{v_2} >=_{\mathtt{lex}} \mathtt{v_3}$. However, Solution 2 does not satisfy the constraint $\mathtt{v_1} >=_{\mathtt{lex}} \mathtt{v_2} >=_{\mathtt{lex}} \mathtt{v_3}$ because the vector $\mathtt{v_1}$ of Solution 2 is $[\mathtt{0,1,1}]$ is not lexicographically greater than the vector $\mathtt{v_2}$, which is $[\mathtt{0,1,0}]$. Hence, Solution 2 is eliminated and the CP solver does not need to explore this solution. Regarding Figure 7.3c, if the individual `1` cannot be assigned to the set variable `A`, the set variable `A` is empty and the individual 1 can be assigned to the set variable `B`. Conversely, if the set variable `A` needs an individual, the individual `1` is always the first choice. □

$$\begin{array}{c|c|c|c|}
 & \mathtt{1} & \mathtt{2} & \mathtt{3} \\ \hline
\mathtt{A} & 1 & 0 & 0 \\ \hline
\mathtt{B} & 0 & 1 & 0 \\ \hline
\mathtt{C} & 0 & 0 & 1 \\ \hline
\end{array}$$

(a) Solution 1

$$\begin{array}{c|c|c|c|}
 & \mathtt{1} & \mathtt{2} & \mathtt{3} \\ \hline
\mathtt{A} & 0 & 0 & 1 \\ \hline
\mathtt{B} & 0 & 1 & 0 \\ \hline
\mathtt{C} & 1 & 0 & 0 \\ \hline
\end{array}$$

(b) Solution 2

$$\begin{array}{c|c|c|c|}
 & \mathtt{1} & \mathtt{2} & \mathtt{3} \\ \hline
\mathtt{A} & 0 & 0 & 0 \\ \hline
\mathtt{B} & 1 & 0 & 0 \\ \hline
\mathtt{C} & 0 & 1 & 0 \\ \hline
\end{array}$$

(c) `A` is empty

Figure 7.3: Assignment of individuals to concepts.

SB4 Upper Triangular matrix of Roles: As described, we encode a binary relation R (role) into an array of set-valued variables `R`. `R` can also be viewed as matrix of Boolean values, where rows and columns are individual names. Figure 7.4 shows an example where the individual `1` is R-related to the individual `2`. In other words, the individual `2` is an R-successor of the individual `1`. Considering that the individuals are introduced according to the number of existential restrictions ($\exists$) and each individual needs at most one R-successor to satisfy an existential restriction, we always can find a solution where a smaller individual is not a R-successor of larger individual, i.e., if a solution states that the individual `1` is an R-successor of the individual `2`, we always can find another solution where the individual `2` is an R-successor of the individual `1` since all individuals are symmetric. Therefore, we can post a symmetry breaking constraint that restricts the matrix to upper triangular form and the main diagonal is zero for every role. Let us describe this through an example.

$$\begin{array}{c|c|c|c|}
 & \mathtt{1} & \mathtt{2} & \mathtt{3} \\ \hline
\mathtt{1} & 0 & 1 & 0 \\ \hline
\mathtt{2} & 0 & 0 & 0 \\ \hline
\mathtt{3} & 0 & 0 & 0 \\ \hline
\end{array}$$

Figure 7.4: Matrix of role R.

Example 7.3.2. Given a TBox $\mathcal{T}$ with axioms $A \sqsubseteq \exists R.B$ and $B \sqsubseteq \exists R.C$, let us check whether the concept A is satisfiable. We now encode $\mathcal{T}$ using the encoding $\mathcal{ELU}^{(\neg)}(f, \Sigma)$2MiniZinc to a MiniZinc model `M`. This example has 3 concepts A, B, C, which are encoded to set variables `A`, `B`, `C`. Since there are 2 existential restrictions, we introduce 3 individuals (1 root individual and 2 individuals for existential restrictions), which are encoded into natural numbers `1`, `2`, `3`. Figure 7.5 shows Solution 1 of `M`, where the individuals `1`, `2`, and `3` are assigned to the set variables `A`, `B`, and `C` respectively, the individual `2` is an R-successor of the individual `1`, and the individual `3` is an R-successor of the individual `2`. If we now flip the concept matrix in Figure 7.5a horizontally, we will get Solution 2 in Figure 7.6. Here the individuals `3`, `2`, and `1` are assigned to the set variables `A`, `B`, and `C` respectively, and the individual `2` is an R-successor of the individual `3` and the individual `1` is an R-successor of the individual `2`. Solution 2 is a symmetric variant of Solution 1.

In order to remove such symmetric solutions, the symmetry breaking constraint that restricts the matrix to upper triangular form and the main diagonal to zero can be added to obtain only Solution 1. $\square$

$$\begin{array}{c|c|c|c|}
 & 1 & 2 & 3 \\ \hline
A & 1 & 0 & 0 \\ \hline
B & 0 & 1 & 0 \\ \hline
C & 0 & 0 & 1 \\ \hline
\end{array}
\qquad
\begin{array}{c|c|c|c|}
 & 1 & 2 & 3 \\ \hline
1 & 0 & 1 & 0 \\ \hline
2 & 0 & 0 & 1 \\ \hline
3 & 0 & 0 & 0 \\ \hline
\end{array}$$

(a) Concept matrix. (b) Role R matrix.

Figure 7.5: Solution 1.

$$\begin{array}{c|c|c|c|}
 & 1 & 2 & 3 \\ \hline
A & 0 & 0 & 1 \\ \hline
B & 0 & 1 & 0 \\ \hline
C & 1 & 0 & 0 \\ \hline
\end{array}
\qquad
\begin{array}{c|c|c|c|}
 & 1 & 2 & 3 \\ \hline
1 & 0 & 0 & 0 \\ \hline
2 & 1 & 0 & 0 \\ \hline
3 & 0 & 1 & 0 \\ \hline
\end{array}$$

(a) Concept matrix. (b) Role R matrix.

Figure 7.6: Solution 2.

The symmetry breaking constraints **SB1**, **SB2**, **SB3** and **SB4** can be combined safely to improve the performance of CP solvers since they are mutually independent and do not interact with each other, except for the combination of **SB3** and **SB4**. As can be seen from Figure 7.5 and Figure 7.6, the assignments of the role matrix depend on the assignments of the concept matrix. Therefore, the ordering of rows of the concept matrix is important.

Example 7.3.3. Let us use the TBox in Example 7.3.2. As mentioned, one of the solutions is shown in Figure 7.5. If we swap rows between the set variables `A` and `B` as shown in Figure 7.7a, we will get an assignment as shown in Figure 7.7. Then if we consider only the symmetry breaking constraint **SB3**, this assignment would be a solution since this assignment satisfies the constraint **SB3**. In contrast, it is not a solution because it does not satisfy the symmetry breaking constraint **SB4**. □

$$\begin{array}{c|c|c|c|}
 & 1 & 2 & 3 \\ \hline
B & 1 & 0 & 0 \\ \hline
A & 0 & 1 & 0 \\ \hline
C & 0 & 0 & 1 \\ \hline
\end{array}
\qquad
\begin{array}{c|c|c|c|}
 & 1 & 2 & 3 \\ \hline
1 & 0 & 0 & 1 \\ \hline
2 & 1 & 0 & 0 \\ \hline
3 & 0 & 0 & 0 \\ \hline
\end{array}$$

(a) Concept matrix. (b) Role R matrix.

Figure 7.7: The problem of the combination between **SB3** and **SB4**.

In order to handle this problem, the ordering of rows of the concept matrix can be determined by the ordering of subsumption between concepts and existential restrictions. This is possible since $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ TBoxes are required to be acyclic and there is no universal restriction in $\mathcal{ELU}^{(\neg)}(f,\Sigma)$. For example, let us again use the TBox containing axioms $A \sqsubseteq \exists R.B$ and $B \sqsubseteq \exists R.C$ in Example 7.3.2. The ordering of rows of the concept matrix is `A`, `B`, and `C`. The solution is given in Figure 7.5.

The experiments in Section 7.4 show that these symmetry breaking constraints have a dramatic effect on the performance of CP solvers.

7.3.3 Search Heuristics

CP solvers, as well as the MiniZinc modelling language, support the declaration of search heuristics, suggesting an order in which the solvers should try assigning variables during the search. The structure of ontologies suggests a search heuristic that attempts to add one individual at a time to each concept in turn (rather than trying to "fill up" a concept before adding individuals to another concept). In addition, care has to be taken if the search heuristic is combined with symmetry breaking constraints for some cases.

The search heuristic can be determined by the concepts that are important for the problem that we are solving. For example, if we want to check the satisfiability of a concept A, the solvers should start assigning the values to the set variable `A` first.

In the case of testing subsumption $A \sqsubseteq B$, we post the symmetry breaking constraints **SB1** and **SB2**. The constraint **SB1** assigns the individual `1` to the set variable `A` of the concept A. We know that B must either be empty or contain the individual `2` since it cannot contain `1` and all other individuals are symmetric. We can therefore use a search heuristic where the solver starts assigning the individual `2` to the set variable `B` first. This heuristic will first try to add 2 to `B`, and in the case of failure the symmetry breaking constraint will force `B` to be empty. CP solvers thus do not need to try other individuals for `B`.

Example 7.3.4. Given a TBox $\mathcal{T}$ containing axioms $A \sqsubseteq \exists R.B$ and $B \sqsubseteq \exists R.C$, let us check whether $A \sqsubseteq B$. We encode $\mathcal{T}$ into a MiniZinc model `M`. Then we add a constraint **`card(A intersect (T diff`** `B)) > 0` for $A \sqsubseteq B$ together with the symmetry breaking constraints **SB1** and **SB2**, resulting in model `M`′. Since there are two existential restrictions, we introduce three individuals (one root individual and two individuals for existential restrictions). Because of **SB1**, CP solvers will assign the individual `1` to `A`. Because of **SB2**, `B` can either be empty or contain the individual `2` since all individuals are symmetric. Then the solver will try assigning the individual `2` to `B` according to the search heuristic above. The solver will easily find that the constraint **`card(A intersect (T diff`** `B)) > 0` is satisfiable and thus $A \not\sqsubseteq B$. If we do not have this search heuristic, the solver may start exploring assignments for the set variable `C` of the concept C first, which is irrelevant for the satisfiability of this problem. □

The experiments in Section 7.4 show that this search heuristic can improve the performance of the CP solvers dramatically for some cases.

7.4 Empirical Evaluation

The following evaluation demonstrates the feasibility of our CP-based approach for handling concrete domain and aggregation in $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. To the best of our knowledge, this is the first implementation of reasoning support for any DL with concrete domains and aggregation, thus no benchmark dataset nor baseline reasoner is available. Hence, evaluation is performed on synthetic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ ontologies.

The reasoning task evaluated here is *Limited Concept Subsumption Checking*, i.e., for a given ontology and two concepts A and B, check whether A is a subclass of B. To check this, as discussed in Section 7.1, we add the negation of the subsumption as an axiom and prove unsatisfiability. The negation is encoded as the constraint **`card(A intersect (T diff`** `B)) > 0`.

We have implemented the encoder `ELUS2MiniZinc` in Java to encode an $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ ontology in (extended) OWL functional syntax into a MiniZinc model. In combination with `ELUS2MiniZinc`, we have selected `chuffed`[1] as CP solver in this evaluation since the evaluation in Section 5.3 showed that `chuffed` outperformed all other solvers. We generated MiniZinc models using `ELUS2MiniZinc`, and then solved the MiniZinc models using `chuffed`. We compare the runtimes of our system over different types of MiniZinc models, with and without symmetry breaking and search heuristic.

[1]`https://github.com/geoffchu/chuffed`

A runtime limit of 100 seconds was used. All results presented in this section have been obtained on a 64bit quad-core `Intel Core i5 3.2GHz` machine, with `8GB` of RAM under ubuntu 16.04. We used MiniZinc version 2.1.3[2]. Note that the results presented in this section include time taken by preprocessing, encoding (taken by `ELUS2MiniZinc`), flattening (translating form MiniZinc to FlatZinc), and solving.

7.4.1 Evaluation Description

Our case study models a hypothetical fitness tracking scenario. It consists of two parts: the first part, a *static* ontology, defines concepts for three basic types of activities, and the second part, a *dynamic* ontology, simulates the incoming stream of fitness activity information as shown in Figures 6.4 and 6.5.

We generated 3000 $\mathcal{ELU}^{(\neg)}(f,\Sigma)$ ontologies to evaluate our approach. These ontologies are similar to the ontology in Figures 6.4 and 6.5, with randomised HR-values of the concepts StateA, StateB and StateC, and randomised hours-value, steps-value and calburn-value of Treadmill, FlexStrider and CrossTrainer. The set of ontologies is divided into three subsets. Each subset has different numbers of states of the stream ontology, which are 3, 5, and 7. We vary the number of activities to test the scalability and feasibility of our approach in handling different numbers of concepts involving concrete domain and aggregations in the stream ontologies. Subsumption checking is performed for each of the 3,000 ontologies.

The random values for HR are between 100 and 220 (a normal range for somebody who is exercising). The random values for hours range from 1 to 3. The steps are calculated from the hours, where the average steps is 4000 per hour. Finally, the calburn is calculated from steps, where the average calories burned is 0.05 per step. The generated test set of 3-states consists of 762 satisfiable ontologies and 238 unsatisfiable ontologies. The 5-states test set consists of 618 satisfiable ontologies and 382 unsatisfiable ontologies. The 7-states test set consists of 562 satisfiable ontologies and 438 unsatisfiable ontologies.

The baseline for our evaluations is (1) a **basic model** which is simply the result of the `ELUS2MiniZinc` encoding, without any symmetry breaking constraints or search heuristics. We compare the performance of this basic model with (2) the basic model plus simple symmetry breaking constraints (**SB1** and **SB2**), (3) the basic model plus symmetry breaking constraints (**SB1**, **SB2**, **SB3**, and **SB4**), (4) the basic model plus search heuristic, and (5) the basic model plus both simple symmetry breaking constraints (**SB1** and **SB2**) and search heuristic (as discussed in Section 7.3). The boxplots for this evaluation contain five series: 1. "basic" shows the results of the basic model, 2. "sb" shows the results of the model with **SB1** and **SB2**, 3. "sb-lex" shows the results of the model with **SB1**, **SB2**, **SB3**, and **SB4**, 4. "s" shows the results of the model with search heuristic, and 5. "ssb" shows the results of the model with search heuristic, **SB1** and **SB2**.

For each pair of ontology and model, the average reasoning time of 100 subsumption checkings was recorded. Hence, for 6 models and 1,000 ontologies per model, a total of 600,000 tests were performed.

7.4.2 Results of the Evaluation

This section presents the results of the evaluation. We compare the performance of our approach `ELU2MiniZinc+chuffed` over the different constraint models described above. The results on all test cases are shown as boxplots, as described in Section 5.3.2.

[2]`http://www.MiniZinc.org`

Results for 3-states ontologies

The overview results on the 3-states ontology test cases are summarised graphically in Figure 7.8. Figure 7.8 shows that all test cases can be solved by our approach within the runtime limitation. The runtime for solving the basic model (basic) is up to 10 seconds. However, the runtime for solving the model with symmetry breaking constraints (sb and sb-lex) is up to 5 seconds. As can be seen, this shows that symmetry breaking constraints have a dramatic effect on the performance of solving.

If we distinguish the test cases with respect to their satisfiability results, we can see that symmetry breaking has a high effect on the performance for unsatisfiable instances as shown in Figure 7.10. On the other hand, symmetry breaking has lesser effect on the performance for satisfiable instances as shown in Figure 7.9. The result of the model with search heuristic (s) is less than 0.5 seconds (Figure 7.8). This shows that search heuristic also has a dramatic impact on the performance of solving, especially for satisfiable instances as shown in Figure 7.9. The performance of search heuristics is even better than the performance of symmetry breaking for satisfiable instances. The search heuristic is crucial for performance, while symmetry breaking has a lesser effect. However, for unsatisfiable instances (Figure 7.10), we get the opposite picture: symmetry breaking speeds up the reasoning dramatically, while the search heuristic alone does not have any clear positive effect. Accordingly, the version with both symmetry breaking and search heuristic (ssb) is the clear winner, and shows almost constant, low runtime. The combination of the two techniques seems crucial to achieve robust performance across satisfiable and unsatisfiable instances.

As can be seen in the results (Figures 7.8, 7.9, and 7.10), interestingly, if we add the symmetry breaking constraints, we get a huge benefit especially in unsatisfiable cases, but not much in satisfiable cases. For the satisfiable cases, we can do something similar. Since `B` is usually not empty in most satisfiable cases, we try to add any individual to `B` first using the search heuristic. The search heuristic works very well in those satisfiable cases, but not in the unsatisfiable cases. However, if we use both together, the results are significantly improved because we can get benefit from both. The empirical evaluation confirms that the symmetry breaking and search heuristic make our approach more effective.

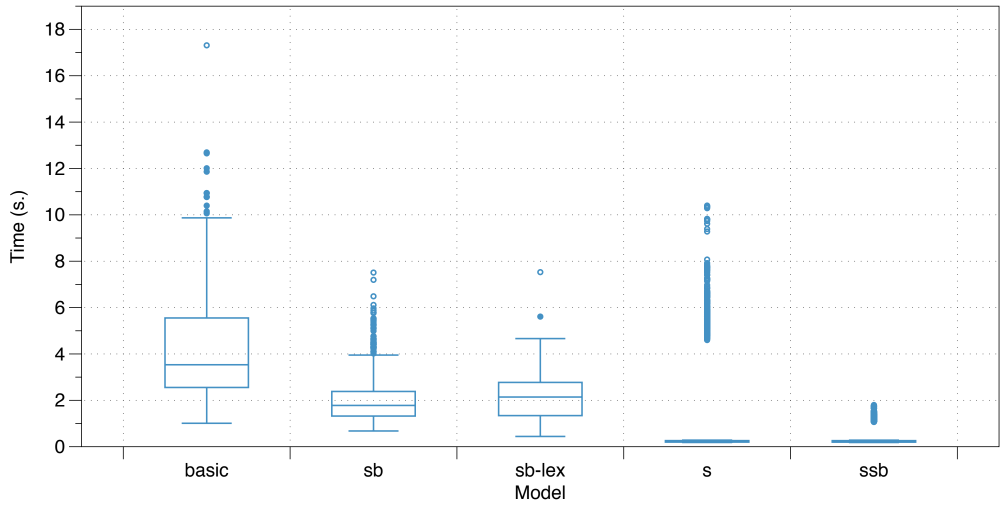


Figure 7.8: Comparison of models with Chuffed: all 3-states ontologies.

We can analyse these experiments further using the number of conflicts in the solving process of `chuffed`. The number of conflicts is the number of failures occurring in the solving process. It also can imply the size of search tree. If the number of conflicts is high,

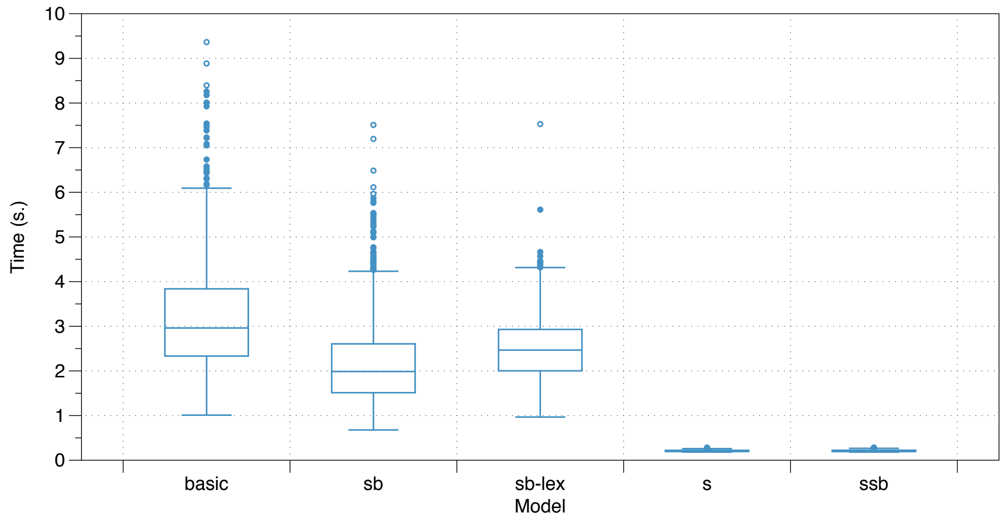


Figure 7.9: Comparison of models with Chuffed: 3-states SAT ontologies.

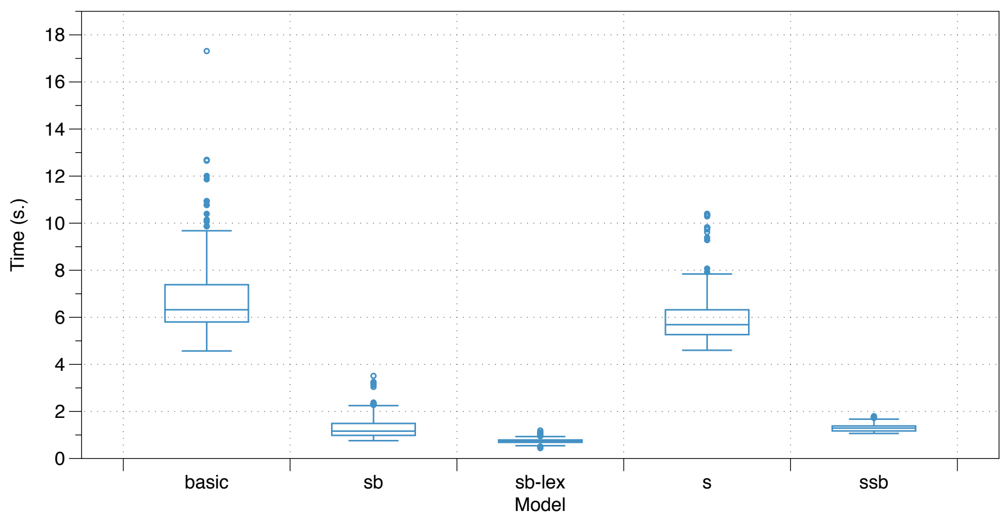


Figure 7.10: Comparison of models with Chuffed: 3-states UNSAT ontologies.

the size of search tree is large. Conversely, if the number of conflicts is low, the size of search tree is small.

Figure 7.11 shows the summary of the number of conflicts of this evaluation. This also shows the reason why the performance of the model with symmetry breaking constraints, the model with search heuristic, and the model with both is better than that of the basic model. As can be seen, the median of the number of conflicts of the model with symmetry breaking constraints is half of the median of the number of conflicts of the basic model. This means that the size of the search tree that the solver need to explore for the model with symmetry breaking constraints is much smaller than that of the basic model. As a result, the performance of the solver over the model with symmetry breaking constraints is better than that of the basic model. The number of conflicts of the model with search heuristic and the model with both symmetry breaking and search heuristic are even lower than that of the model with symmetry breaking, and almost constant. Figures 7.12 and 7.13 show that the number of conflicts of unsatisfiable cases is lower than that of satisfiable cases in general. Again, with this measurement, we get huge advantages especially in unsatisfiable cases, but not much in satisfiable cases if we add the symmetry breaking constraints. For unsatisfiable cases (Figure 7.13), the symmetry breaking significantly reduces the search tree size that needs to be explored (six times lower than the number of conflicts of the basic model). This means that symmetry breaking manages to get rid of a lot of symmetric sub search trees. On the other hand, for satisfiable cases (Figure 7.12), the number of conflicts of the model with search heuristic is very low and almost constant since the search heuristic guides the solver with the appropriate variable order. Again, the version with both symmetry breaking and search heuristic is the clear winner. It significantly reduces the size of the search tree for both satisfiable and unsatisfiable cases.

In conclusion, the runtime and number of conflicts have a positive correlation. For the 3-states ontology test cases, the model with both symmetry breaking and search heuristic obtains the best performance among the other models since we can get benefits from both symmetry breaking and search heuristic.

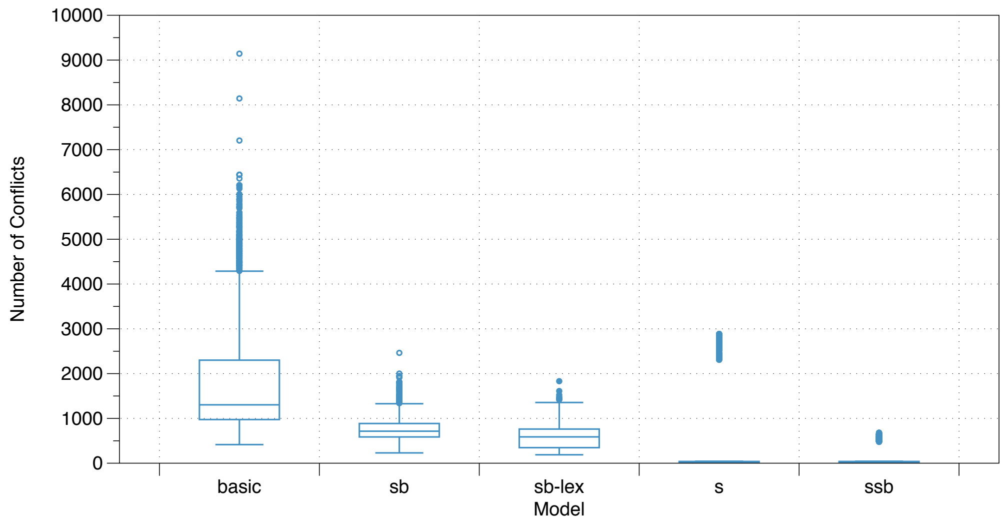


Figure 7.11: Number of conflicts comparison of models with Chuffed: all 3-states ontologies.

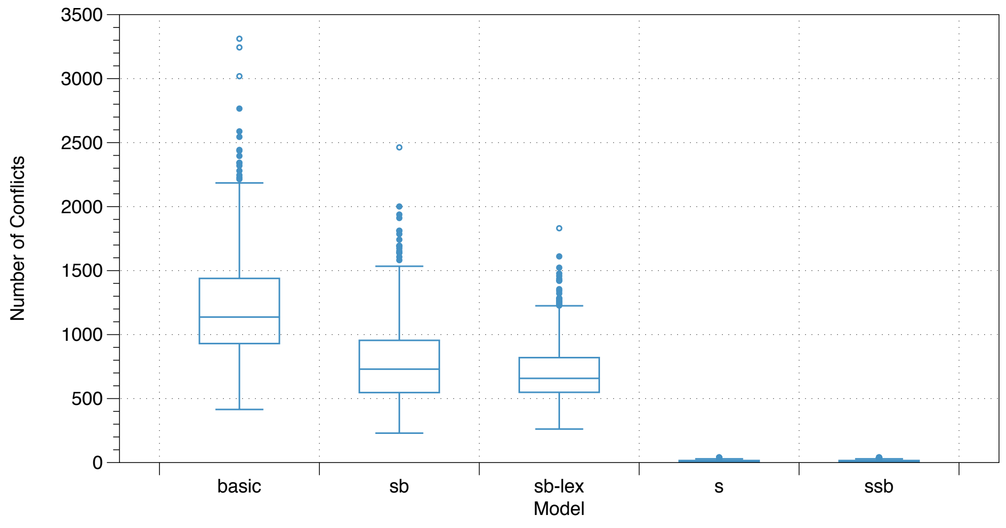


Figure 7.12: Number of conflicts comparison of models with Chuffed: 3-states SAT ontologies.

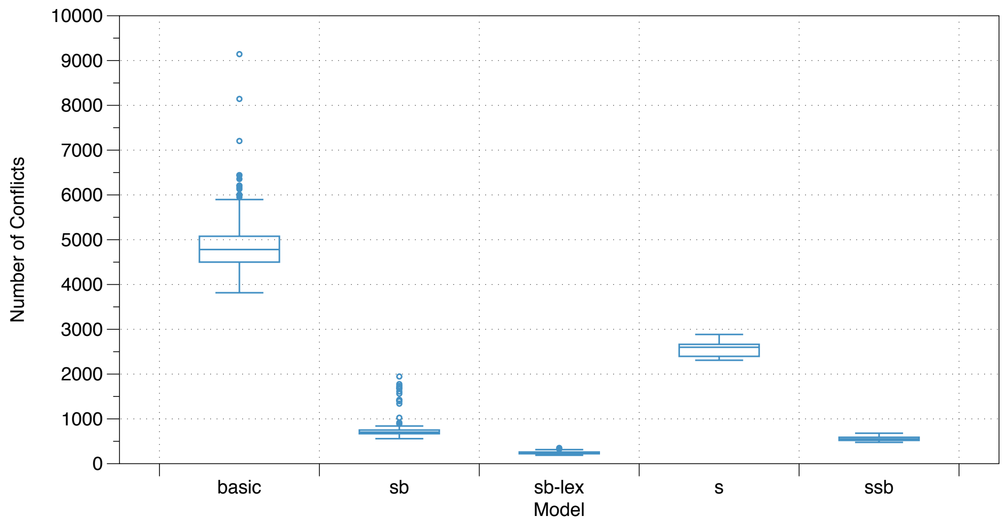


Figure 7.13: Number of conflicts comparison of models with Chuffed: 3-states UNSAT ontologies.

Results for 5-states ontologies

The overview results on the 5-states ontology test cases are summarised graphically in Figure 7.14. Figure 7.14 shows that all test cases can be solved by our approach within the runtime limitation. The runtime for solving the basic model (basic) is up to 80 seconds. The runtime for solving the model with symmetry breaking constraints (sb) is up to 15 seconds. The runtime for solving the model with lexicographical ordering constraints (sb-lex) is up to 10 seconds. As can be seen, this shows that symmetry breaking constraints have a large effect on the performance of solving. The results of the evaluation also show that the sb and sb-lex models outperform the other models for both satisfiable and unsatisfiable cases (Figure 7.15 and 7.16).

Now, if we consider the test cases with respect to their satisfiability results separately, it can be seen that symmetry breaking has a big effect on the performance for both unsatisfiable instances as shown in Figure 7.16 and satisfiable instances as shown in Figure 7.16. Considering the median of each the model, the versions sb and sb-lex are almost 45 times faster than the basic model for both satisfiable and unsatisfiable cases. On the other hand, the models with search heuristic (s), as well as symmetry breaking combined with search heuristic (ssb), do not work effectively for either satisfiable or unsatisfiable cases. This is because the search heuristic that we used may guide the solver with an inappropriate variable ordering or contradict the symmetry breaking constraints. This leads the solver far from the solution. All of the unsatisfiable cases time out in this case.

Next, we analyse this evaluation using the number of conflicts occurring in the solving process. Figure 7.17 shows the summary of the number of conflicts of the evaluation of the 5-states ontology test cases. As can be seen, the distributions of number of conflicts of the models sb and sb-lex are much smaller than those of the basic model. The maximum numbers of conflicts of the models sb and sb-lex are almost seven times lower than the maximum number of conflicts of the basic model. This means that the sizes of the search trees that the solver needs to explore for the models sb and sb-lex are smaller than that of the basic model. As a result, the performance of the solver over the models sb and sb-lex is better than that of the basic model. In contrast, the number of conflicts of the model with search heuristic and the model with both symmetry breaking and search heuristic are very high. Note that the numbers of conflicts of the models s and ssb are assumed as

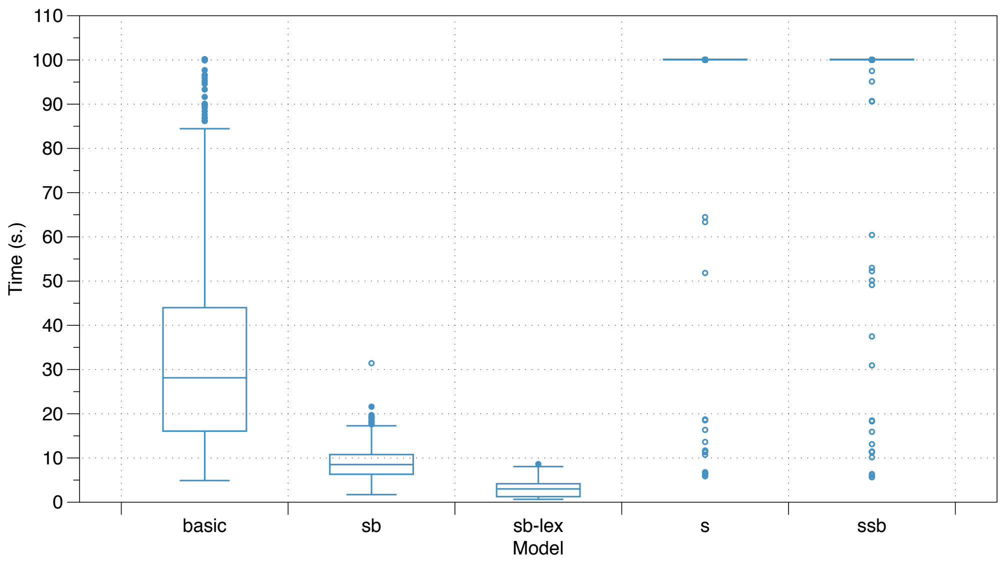


Figure 7.14: Comparison of models with Chuffed: all 5-states ontologies.

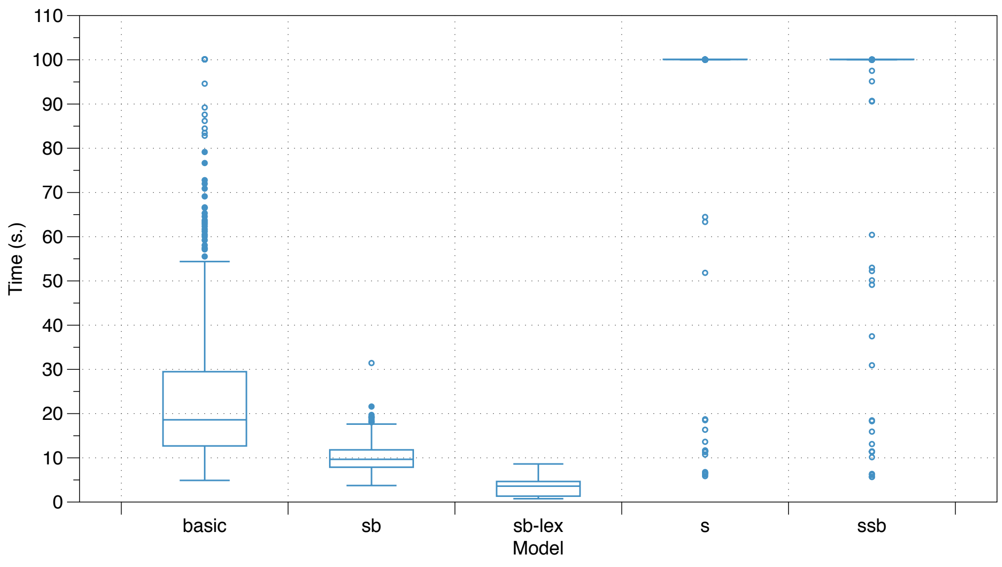


Figure 7.15: Comparison of models with Chuffed: 5-states SAT ontologies.

the highest number (48,000) of all test cases since these two models time out. The actual numbers of conflicts of the model s and ssb may be much higher than these.

Figures 7.18 and 7.19 show that the number of conflicts of unsatisfiable cases (the median number is around 19,000 conflicts for the basic model) is higher than that of satisfiable cases (the median number is around 6,000 conflicts for the basic model) in general. With this measurement, we get large advantages from symmetry breaking in both satisfiable and unsatisfiable cases. For unsatisfiable cases (Figure 7.19), considering the median of the number of conflicts, the symmetry breaking significantly reduces the size of the search tree that needs to be explored (ten times lower than the number of conflicts of the basic model). This means that symmetry breaking eliminates a lot of symmetric sub search trees. Similarly, for satisfiable cases (Figure 7.18), the median number of conflicts of the models sb and sb-lex are twice as low as that of the basic model. The numbers of conflicts of the models s and ssb are very high compared to that of the basic model for both satisfiable and unsatisfiable cases.

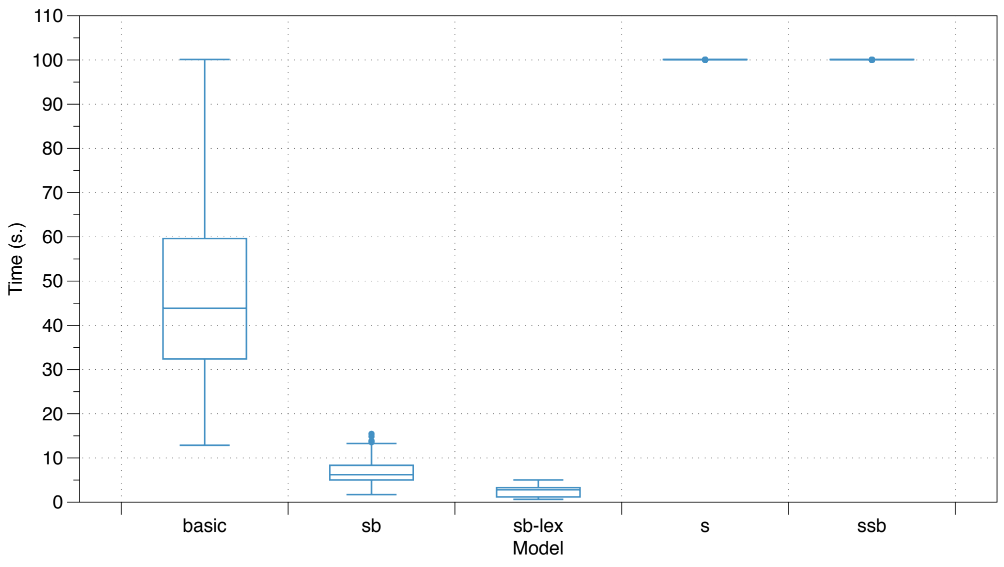


Figure 7.16: Comparison of models with Chuffed: 5-states UNSAT ontologies.

In conclusion, the runtime and number of conflicts have a positive correlation. For the 5-states ontology test cases, the model sb-lex obtains the best performance since the lexicographical ordering constraints significantly reduce the size of search tree.

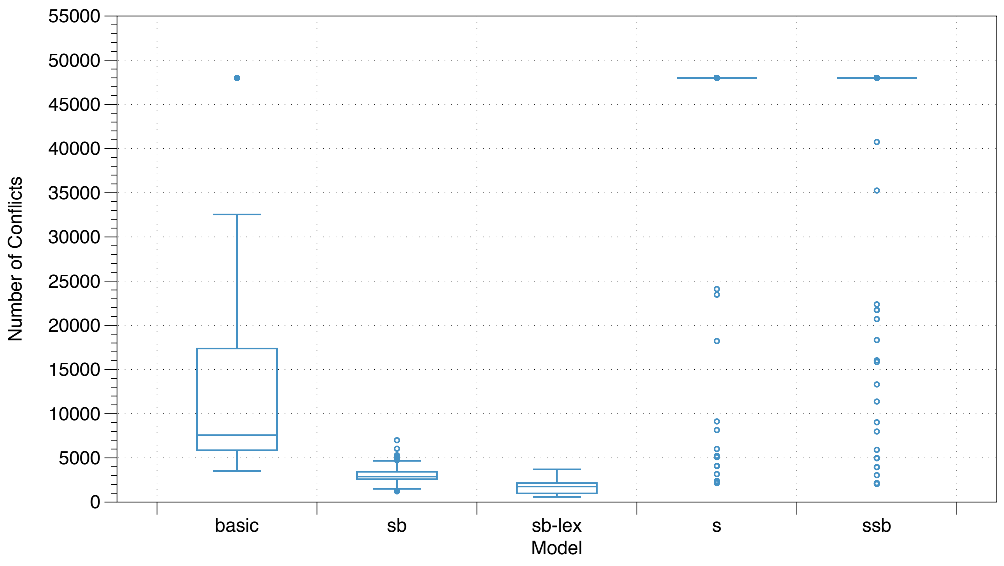


Figure 7.17: Number of conflicts comparison of models with Chuffed: all 5-states ontologies.

Results for 7-states ontologies

The overview results on the 7-states ontology test cases are summarised graphically in Figure 7.20. It shows that the most of the models, basic, s, and ssb, time out since these test cases contain many concepts involving concrete domain and aggregations. However, the models with symmetry breaking constraints (sb and sb-lex) are still solvable within the time limit. As can be seen, this shows that the symmetry breaking constraints, especially the lexicographical ordering constraints, have a dramatic effect on the performance of

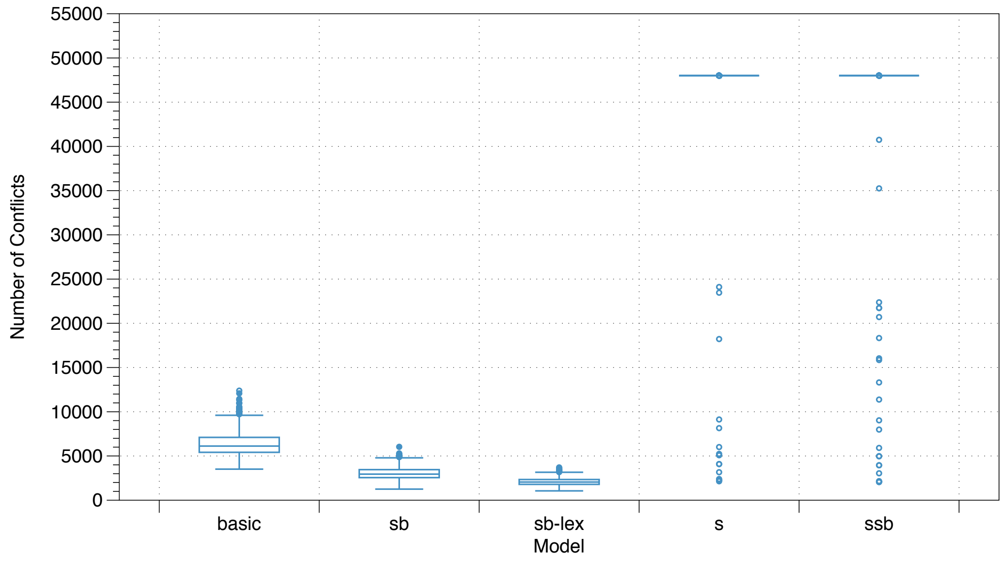


Figure 7.18: Number of conflicts comparison of models with Chuffed: 5-states SAT ontologies.

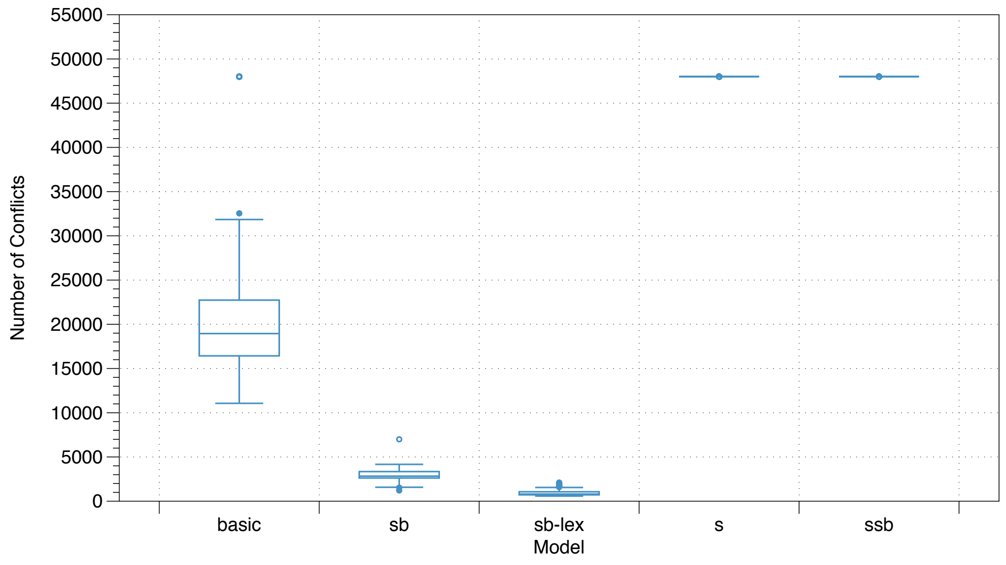


Figure 7.19: Number of conflicts comparison of models with Chuffed: 5-states UNSAT ontologies.

solving. In addition, the results of the evaluation show that the sb-lex model outperforms the other models for both satisfiable and unsatisfiable cases (Figure 7.21 and 7.22).

Next, if we consider the test cases with respect to their satisfiability results separately, it can be seen that the lexicographical ordering symmetry breaking still has a high effect on the performance for both unsatisfiable instances as shown in Figure 7.22 and satisfiable instances as shown in Figure 7.22. Considering the median of each the model, the runtime of the sb-lex version is almost 4 times faster than the basic model for both satisfiable and unsatisfiable cases. The runtime of the sb model is a bit faster than the basic model for satisfiable cases. On the other hand, the models with search heuristic (s), and both symmetry breaking and search heuristic (ssb) do not work effectively for both satisfiable and unsatisfiable cases. The reason is similar to the one of the 5-states ontology test cases. The search heuristic that we used may guide the solver with an inappropriate variable ordering or contradict the symmetry breaking constraints. All s and ssb model test cases time out.

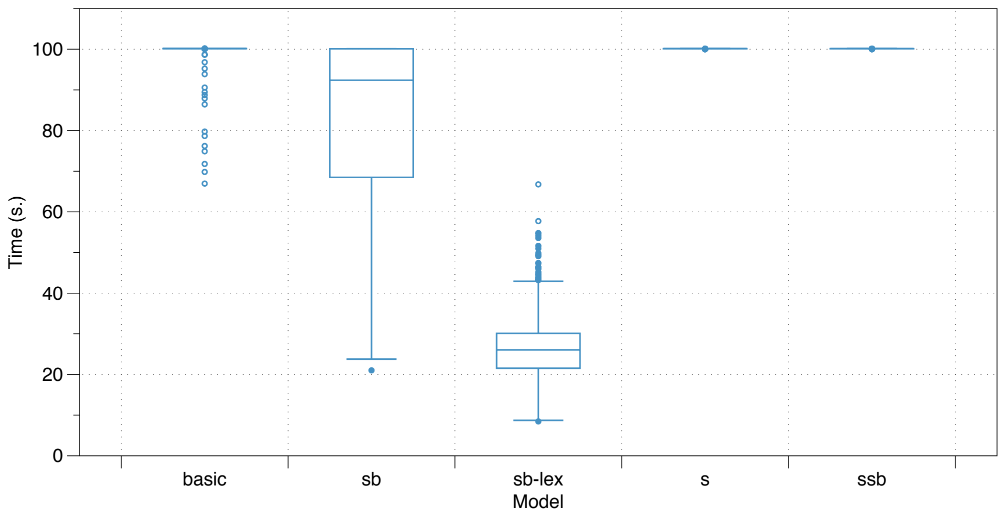


Figure 7.20: Comparison of models with Chuffed: all 7-states ontologies.

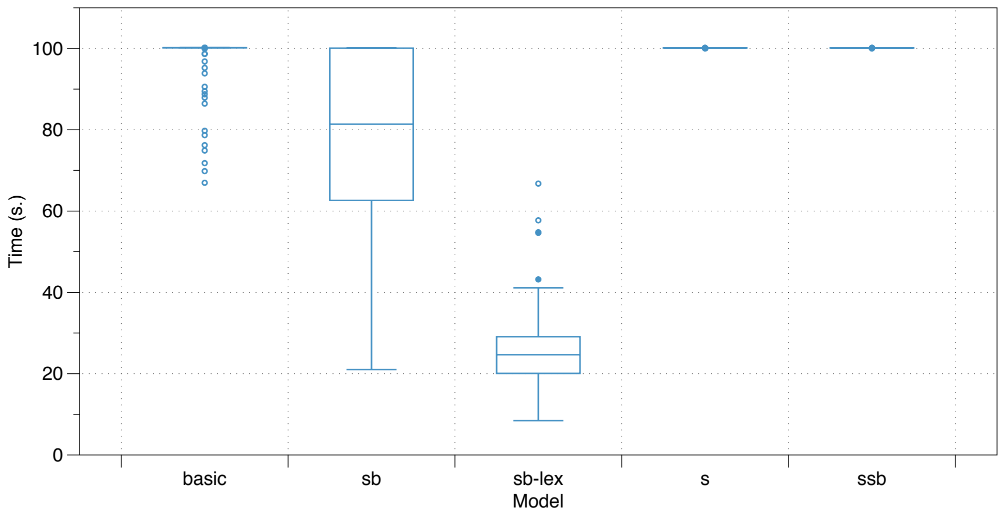


Figure 7.21: Comparison of models with Chuffed: 7-states SAT ontologies.

Next, we analyse this evaluation using the number of conflicts occurring in the solving process. Figure 7.23 shows the summary of the number of conflicts of the evaluation of the 7-states ontology test cases. As can be seen, the distribution of number of conflicts of the model sb-lex is much smaller than that of the basic model. The maximum number of conflicts of the model sb-lex is almost 4 times lower than the maximum number of conflicts of the basic model. This means that the size of the search tree that the solver needs to explore for the model sb-lex is smaller than that of the basic model. Thus, the performance of the solver over the model sb-lex is better than that of the basic model. In contrast, the number of conflicts of the models, sb, s, and ssb, are very high. Note that some numbers of conflicts of the models, sb, s, and ssb, are assumed as the highest number (69,000) of all test cases since the runtimes of these three models are beyond the time limit. The actual numbers of conflicts of the models, sb, s, and ssb, may be much higher than these for some cases.

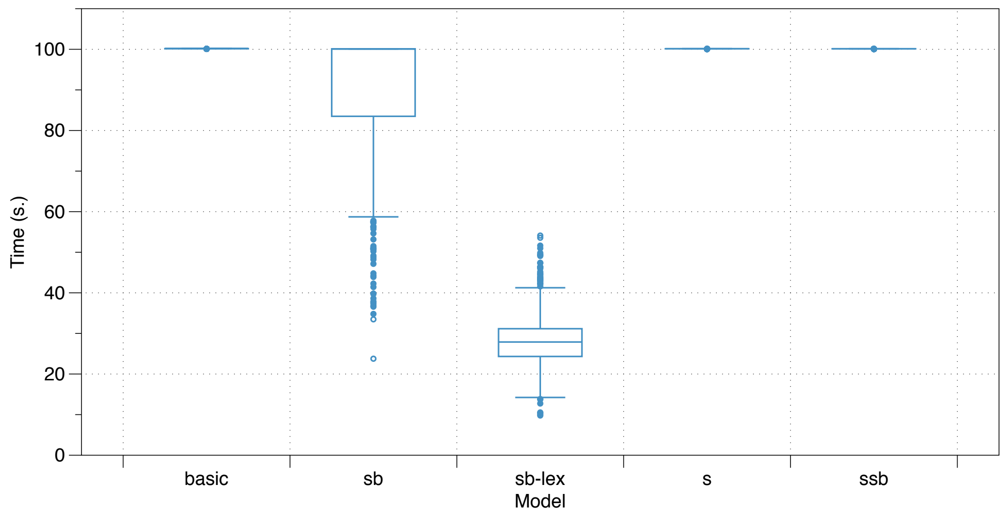


Figure 7.22: Comparison of models with Chuffed: 7-states UNSAT ontologies.

Figures 7.24 and 7.25 show that the numbers of conflicts of unsatisfiable cases of the models sb and sb-lex are lower than those of satisfiable cases in general. With this measurement, we get a huge advantage from the lexicographical ordering symmetry breaking in both satisfiable and unsatisfiable cases. For unsatisfiable cases (Figure 7.25), considering the median of the number of conflicts, the lexicographical ordering constraints significantly reduce the size of the search tree (5 times lower than the number of conflicts of the basic model). This means that the lexicographical ordering gets rid of a lot of symmetric sub search trees. Similarly, for satisfiable cases (Figure 7.24), the median number of conflicts of the model sb-lex is 7 times as low as that of the basic model. The numbers of conflicts of the models s and ssb are very high compared to that of the basic model for both satisfiable and unsatisfiable cases.

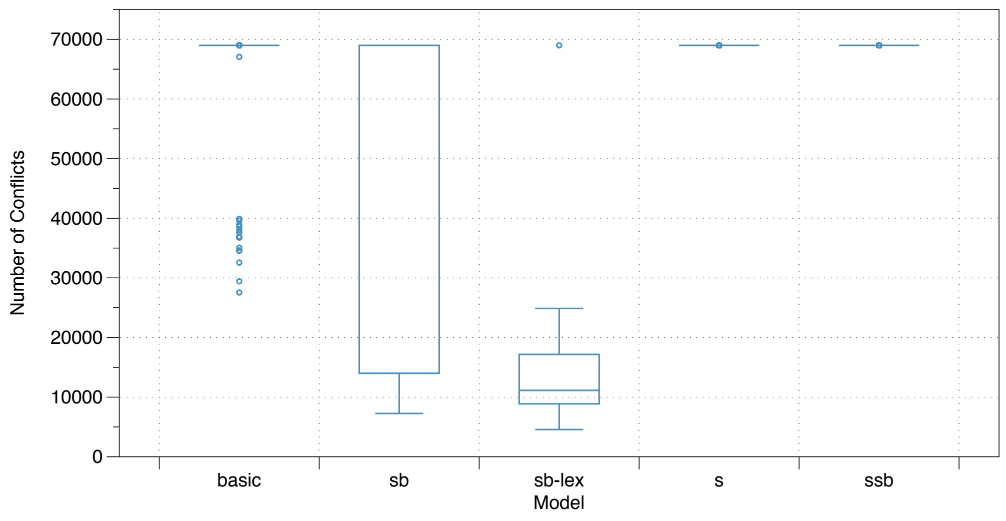


Figure 7.23: Number of conflicts comparison of models with Chuffed: all 7-states ontologies.

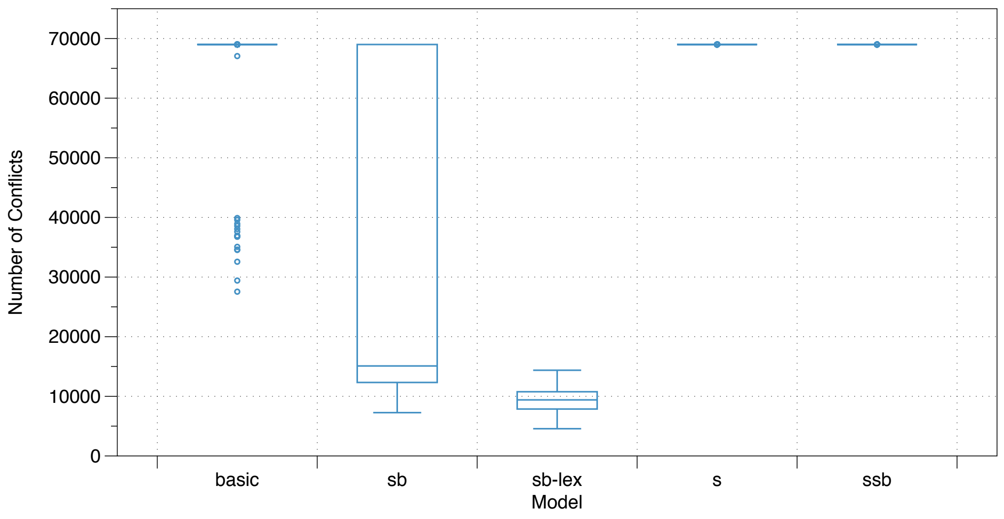


Figure 7.24: Number of conflicts comparison of models with Chuffed: 7-states SAT ontologies.

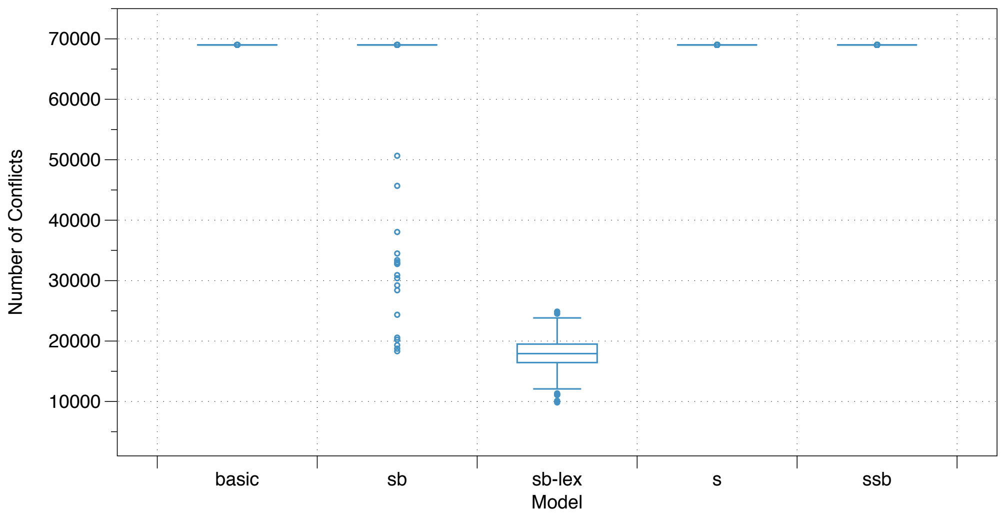


Figure 7.25: Number of conflicts comparison of models with Chuffed: 7-states UNSAT ontologies.

Summary of the results

Then we analyse this evaluation using the ratio of the runtime of the models sb, s, ssb, and basic to the runtime of the model sb-lex. Figure 7.26 shows the runtime ratio for all ontologies (including 3-states, 5-states, and 7-states) where the model sb-lex is the base line. The runtime ratio is presented as log scale since the range of data is large. It can be clearly seen that the lexicographical ordering symmetry breaking constraint helps the solver to perform solving faster than the other models for most of ontologies. The outliers also show that the runtime ratio of the other models to sb-lex is very high (around 100) for some ontologies. As a consequence, in general, the lexicographical ordering symmetry breaking constraint (model sb-lex) works very well for most of ontologies. For some ontologies, the runtime ratio of models s and ssb is lower than 1 since the search heuristic (model s) and the combination of symmetry breaking and search heuristic (model ssb) work very well for the ontologies with a small number of states.

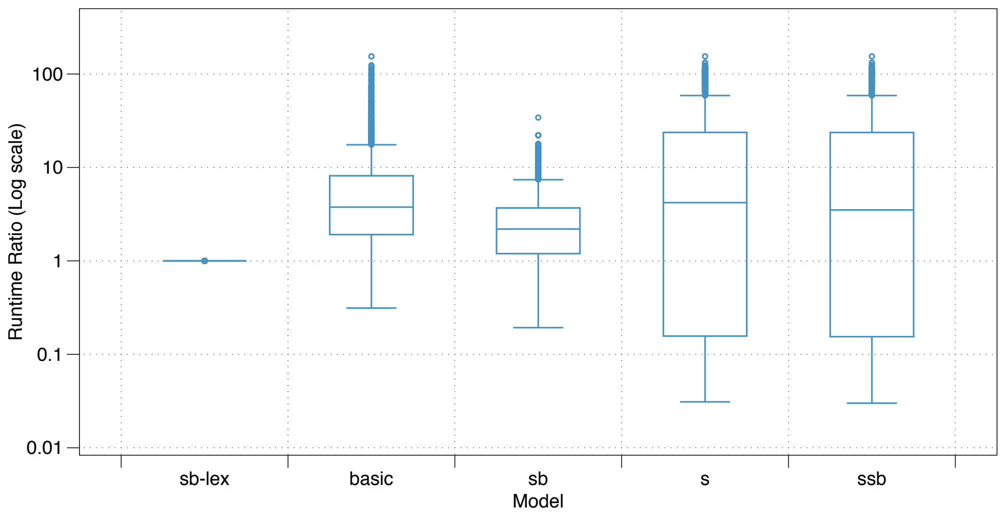


Figure 7.26: Runtime ratio for all ontologies.

We further analyse the ratio of the runtime of the models sb, s, ssb, and basic to the runtime of the model sb-lex using geometric mean as shown in Table 7.1. The geometric mean indicates the central tendency of the set of numbers by using their product. Since we deal with ratio, the geometric mean is reasonable measurement to use. Considering the geometric mean of the runtime ratio for all ontologies, it can be seen that the runtime of model sb-lex is 4 times faster than that of the basic model. The runtime of model sb-lex is around 2 times faster than that of models sb and ssb. The runtime of model sb-lex is around 3 times faster than that of model s. This is evidence that the lexicographical ordering constraint can help to improve our approach effectively.

Table 7.1: Geometric mean of runtime ratio for all ontologies.

sb-lex	basic	sb	s	ssb
1	4.180	2.132	3.225	2.852

Now, we analyse this evaluation using the ratio of the number of conflicts of the models sb, s, ssb, and basic to the number of conflicts of the model sb-lex. The number of conflicts ratio for all ontologies, where the model sb-lex is the base line, is presented in Figure 7.27. The number of conflicts ratio is presented as log scale since the range of data is large. It can be easily seen that the lexicographical ordering symmetry breaking constraint helps the solver to reduce the search space more than the other models for most of ontologies. The outliers also show that the number of conflicts ratio of the other models to sb-lex is very high (almost 100) for some ontologies. For some test cases, the number of conflicts ratio of models s and ssb is lower than 1 because the search heuristic (model s) and the combination of symmetry breaking and search heuristic (model ssb) work very well for the ontologies with a small number of states.

Table 7.2 shows the geometric mean of the ratio of the number of conflicts of the models sb, s, ssb, and basic to the runtime of the model sb-lex. From The table shows that the number of conflicts of model sb-lex is almost 5 times lower than that of the basic model. The number of conflicts of model sb-lex is around 2 times lower than that of models sb, s , and ssb. This confirms that the lexicographical ordering constraint can help to improve our approach effectively in general.

In summary, the results of all our test case sets show that symmetry breaking is more powerful than the given search heuristic in general for our problems. Symmetry breaking

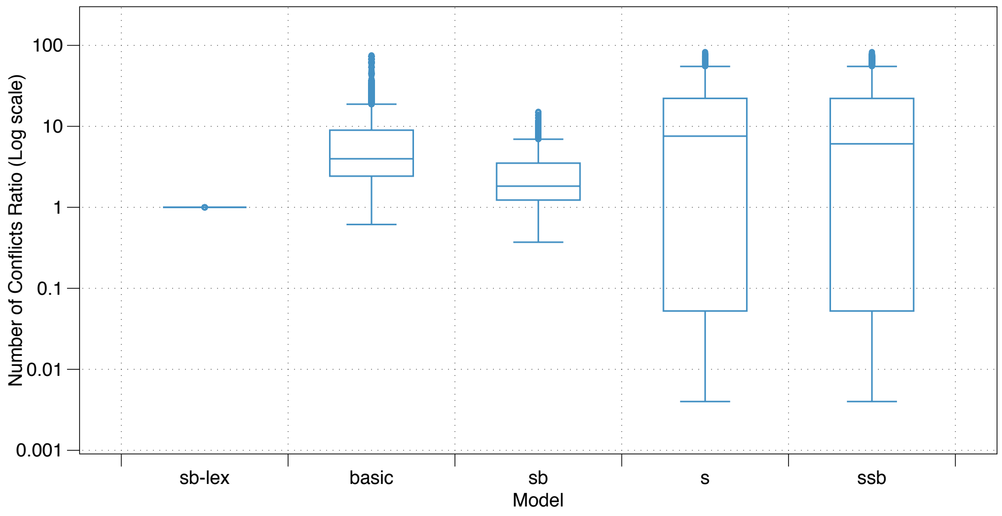


Figure 7.27: Number of conflicts ratio for all ontologies.

Table 7.2: Geometric mean of number of conflicts ratio for all ontologies.

sb-lex	basic	sb	s	ssb
1	4.913	2.078	2.371	2.098

can dramatically reduce the size of search tree and improve the solving process. It is interesting to investigate further which search heuristic may be useful for our problems and exploit more symmetry breaking constraints.

From the evaluation in Sections 5.3 and 7.4, if we carefully analyse the structure of the dataset, we can find that the size of ontologies does not affect the performance of our approach. However, if ontologies contain many concepts involving aggregations, the performance of our approach is affected. This confirms that our approach is efficient for the existing DLs, i.e, $\mathcal{ALC}$ and it is feasible and effective for our logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. Most of the $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ ontologies in the test case are solvable by our approach.

Chapter 8

Conclusions

Description Logics with concrete domains and aggregation can be used to provide precise definitions of knowledge involving actual numeric values. Despite their usefulness, development of such Description Logics has been very limited, mainly due to the difficulty in balancing decidability and expressivity, and difficulty in extending existing tableau-based algorithms.

The first goal of this thesis aims at developing a novel Description Logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ that supports concrete domains, aggregations for modelling knowledge base (ontology). In addition, we have shown that concept satisfiability of $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ is NP-COMPLETE. This is achieved through some reasonable syntactic restrictions on $\mathcal{ELU}^{(\neg)}(f, \Sigma)$.

The second goal of this thesis aims at developing new techniques for reasoning in Description Logics based on a sound and complete encoding into MiniZinc, which allows us to exploit efficient Constraint Programming technology, for $\mathcal{ALC}$ and $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. To the best of our knowledge this is the first implementation of reasoning support for a Description Logic with concrete domains and aggregation. Our empirical analysis shows that our approach is efficient for some $\mathcal{ALC}$ ontologies and it is feasible and effective for $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ ontologies.

This chapter presents a summary of the contributions and some further research questions that are worth exploring.

8.1 Summary and Main Contributions

Addressing the first goal, we investigated language constructs of Description Logics in order to obtain a decidable Description Logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ that supports concrete domains and aggregations for modelling knowledge bases. We defined the syntax (including the extended OWL functional syntax) and semantics of $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. We presented inference problems, *concept satisfiability*, *limited concept subsumption*, and *consistency checking*, that can arise from $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. We proved that $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ is decidable and the concept satisfiability, limited concept subsumption, and consistency checking tasks of $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ are NP-COMPLETE.

Addressing the second goal, the second part of this thesis presented an approach that exploits CP techniques in automated reasoning for Description Logics.

We have proposed the set-based encoding scheme in order to translate ontologies into MiniZinc constraint models in a very high-level and succinct way for the most well-known Description Logic $\mathcal{ALC}$ and our Description Logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. Such constraint models can be solved by modern CP solvers, supporting reasoning services *concept satisfiability*, *limited concept subsumption*, and *consistency checking*. In addition, we investigated some optimisation techniques in order to improve the efficiency of our encoding approach. These optimisation techniques includes *symmetry breaking*, especially lexicographical ordering symmetry breaking constraints, and *search heuristic*.

We have developed two different prototype tools, which implement the above proposed encoding scheme. These tools were evaluated in our extensive empirical analysis. We have shown that the performance of our approach is competitive to and sometimes better than that of the state-of-the-art tableau-based reasoners for some $\mathcal{ALC}$ ontologies, especially ontologies that contain a lot of disjunctions. However, our approach does not perform very well on ontologies that contain long chains of existential restrictions (i.e., the Tableaux'98 benchmark). Furthermore, we have shown that our approach is able to effectively handle reasoning tasks for $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. We also show that a number of techniques proposed by CP research can be easily reused and adapted for solving DL reasoning problems, significantly improving reasoning performance. In addition, our approach allows us to take advantages of current and future improvements in CP solvers for free without much additional effort or modification in particular for concrete domain and aggregations.

8.2 Future Research

The work presented in this thesis can be further explored and extended in a number of possible directions.

Extending $\mathcal{ELU}^{(\neg)}(f, \Sigma)$. The novel Description Logic $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ proposed in this thesis retains decidability due to some syntactic restrictions. The feasibility of relaxing certain syntactic restrictions on the logic without sacrificing reasoning efficiency is one direction to be further investigated. In addition, it is interesting to extend this logic with some additional language constructs such as qualified number restrictions and universal restrictions. Moreover, it is interesting to investigate integers as concrete domain.

Tightening a bounded universe. As can be seen from the evaluation, the performance of our approach depends on the number of individuals for MiniZinc models. The performance of our approach is poor for some cases since the number of individuals calculated is very large. It is interesting to investigate an approach to tighten the number of individuals for MiniZinc models.

Extending the Encoding Scheme. The encoding scheme proposed in this thesis supports $\mathcal{ALC}$ and its sub-logics, and $\mathcal{ELU}^{(\neg)}(f, \Sigma)$ with respect to acyclic TBoxes. It is possible to extend this approach to support more expressive Description Logics such as Description Logics with qualified number restrictions (e.g., $\mathcal{ALCQ}$) and Description Logics with role characteristics (i.e., role hierarchies and transitive role). In addition, this approach can be investigated further to support general TBoxes.

Extending Optimisation Techniques. As shown in the evaluation, symmetry breaking, especially lexicographical ordering symmetry breaking constraints and search heuristic help to improve the performance of our approach significantly. It is interesting to study CP solving and optimisation techniques to improve efficiency and scalability. For example, symmetry breaking can be further investigated. In addition, an appropriate search heuristic can be further studied. Moreover, blocking techniques are worth to be studied and added to the pre-calculation of number of individuals in order to support cyclic TBoxes.

Extending Support for Reasoning Services. We have shown that our approach can support the reasoning services consistency checking, concept satisfiability, and concept subsumption checking. It is possible to extend our approach to support more reasoning services such as classification and instance checking.

Appendix A

MiniZinc constraints of the stream ontology in Figure 6.5

This appendix presents the complete MiniZinc constraint model of the stream ontology in Figure 6.5 in Section 6.3.2.

```
constraint forall(i in StateA)(
 card(excercise[i] ∩ Treadmill) >= 1 ∧ (id[i] = 1) ∧ (HR[i] = 100)
 hours[i] = s_sum_hours[i] +
        sum (j in exercise[i]) (hours[i]) ∧
 calburn[i] = s_sum_calburn[i] +
        sum (j in exercise[i]) (calburn[i]) ∧
 steps[i] = s_sum_steps[i] +
        sum (j in exercise[i]) (steps[i]) ∧
 (card(has[i]) + s_count_has_hours[i] +
        bool2int(b_max_has_hours[i]) +
        bool2int(b_min_has_hours[i]) = 1) ∧
 (card(has[i]) + s_count_has_calburn[i] +
        bool2int(b_max_has_calburn[i]) +
        bool2int(b_min_has_calburn[i]) = 1) ∧
 (card(has[i]) + s_count_has_steps[i] +
        bool2int(b_max_has_steps[i]) +
        bool2int(b_min_has_steps[i]) = 1) );                    (C5)
constraint forall(i in StateB)(
 card(excercise[i] ∩ Treadmill) >= 1 ∧ (id[i] = 2) ∧ (HR[i] = 110)
 hours[i] = s_sum_hours[i] +
        sum (j in exercise[i]) (hours[i]) ∧
 calburn[i] = s_sum_calburn[i] +
        sum (j in exercise[i]) (calburn[i]) ∧
 steps[i] = s_sum_steps[i] +
        sum (j in exercise[i]) (steps[i]) ∧
 (card(has[i]) + s_count_has_hours[i] +
        bool2int(b_max_has_hours[i]) +
        bool2int(b_min_has_hours[i]) = 1) ∧
 (card(has[i]) + s_count_has_calburn[i] +
        bool2int(b_max_has_calburn[i]) +
        bool2int(b_min_has_calburn[i]) = 1) ∧
 (card(has[i]) + s_count_has_steps[i] +
        bool2int(b_max_has_steps[i]) +
        bool2int(b_min_has_steps[i]) = 1) );                    (C6)
```

```
constraint forall(i in StateC)(
 card(excercise[i] ∩ Treadmill) >= 1 ∧ (id[i] = 3) ∧ (HR[i] = 128)
 hours[i] = s_sum_hours[i] +
        sum (j in exercise[i]) (hours[i]) ∧
 calburn[i] = s_sum_calburn[i] +
        sum (j in exercise[i]) (calburn[i]) ∧
 steps[i] = s_sum_steps[i] +
        sum (j in exercise[i]) (steps[i]) ∧
 (card(has[i]) + s_count_has_hours[i] +
        bool2int(b_max_has_hours[i]) +
        bool2int(b_min_has_hours[i]) = 1) ∧
 (card(has[i]) + s_count_has_calburn[i] +
        bool2int(b_max_has_calburn[i]) +
        bool2int(b_min_has_calburn[i]) = 1) ∧
 (card(has[i]) + s_count_has_steps[i] +
        bool2int(b_max_has_steps[i]) +
        bool2int(b_min_has_steps[i]) = 1) );                    (C7)
constraint forall(i in T)(
 i in DailyAggregate <->
 (card(has[i] ∩ StateA) >= 1 ∧
  card(has[i] ∩ StateB) >= 1 ∧
  card(has[i] ∩ StateC) >= 1 ∧
  HR[i] = max (j in has[i]) (HR[j]) ∧
  hours[i] = s_sum_hours[i] +
        sum (j in exercise[i]) (hours[i]) ∧
  calburn[i] = s_sum_calburn[i] +
        sum (j in exercise[i]) (calburn[i]) ∧
  steps[i] = s_sum_steps[i] +
        sum (j in exercise[i]) (steps[i]) ∧
 (card(has[i]) + s_count_has_hours[i] +
        bool2int(b_max_has_hours[i]) +
        bool2int(b_min_has_hours[i]) = 3) ∧
 (card(has[i]) + s_count_has_calburn[i] +
        bool2int(b_max_has_calburn[i]) +
        bool2int(b_min_has_calburn[i]) = 3) ∧
 (card(has[i]) + s_count_has_steps[i] +
        bool2int(b_max_has_steps[i]) +
        bool2int(b_min_has_steps[i]) = 3) );                    (C8)
```

Vita

Publications arising from this thesis include:

Sawangphol, W., Li, Y.-F., and Tack, G. (2016), CP4DL: Constraint-based Reasoning for Expressive Description Logics. In *The fifteenth International Workshop on Constraint Modelling and Reformulation (ModRef 2016).* Toulouse, France.

Permanent Address:	Faculty of Information Technology
	Monash University
	Australia

This thesis was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Glenn Maughan and modified by Dean Thompson and David Squire of Monash University.

References

Areces, C., Bouma, W., & de Rijke, M. (1999). Description Logics and Feature Interaction. In *Proceedings of the 1999 International Workshop on Description Logics (DL'99), Linköping, Sweden, July 30 - August 1, 1999.* Retrieved from `http://ceur-ws.org/Vol-22/areces.ps`

Artale, A., Calvanese, D., Kontchakov, R., & Zakharyaschev, M. (2009). The dl-lite family and relations. *J. Artif. Intell. Res. (JAIR)*, *36*, 1–69. Retrieved from `http://dx.doi.org/10.1613/jair.2820` doi: 10.1613/jair.2820

Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., … Sherlock, G. (2000, May 01). Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature genetics*, *25*(1), 25–29. Retrieved from `http://dx.doi.org/10.1038/75556` doi: 10.1038/75556

Baader, F. (1991). Augmenting concept languages by transitive closure of roles: an alternative to terminological cycles. In *Proceedings of the 12th international joint conference on artificial intelligence - volume 1* (pp. 446–451). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved from `http://dl.acm.org/citation.cfm?id=1631171.1631238`

Baader, F. (2000). Tableau Algorithms for Description Logics. In *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2000, St Andrews, Scotland, UK, July 3-7, 2000, Proceedings* (pp. 1–18). Retrieved from `http://dx.doi.org/10.1007/10722086_1` doi: 10.1007/10722086_1

Baader, F. (2003). Terminological Cycles in a Description Logic with Existential Restrictions. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003* (pp. 325–330). Retrieved from `http://ijcai.org/Proceedings/03/Papers/048.pdf`

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the $\mathcal{EL}$ Envelope. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005* (pp. 364–369). Retrieved from `http://ijcai.org/Proceedings/05/Papers/0372.pdf`

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2003). *The Description Logic Handbook: Theory, Implementation, and Applications.* New York, NY, USA: Cambridge University Press.

Baader, F., & Hanschke, P. (1991). A Scheme for Integrating Concrete Domains into Concept Languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1* (pp. 452–457). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved from `http://dl.acm.org/citation.cfm?id=1631171.1631239`

Baader, F., & Hanschke, P. (1992). Extensions of Concept Languages for a Mechanical Engineering Application. In *GWAI-92: Advances in Artificial Intelligence, 16th German Conference on Artificial Intelligence, Bonn, Germany, August 31 - September 3, 1992, Proceedings* (pp. 132–143). Retrieved from `http://dx.doi.org/10.1007/BFb0018999` doi: 10.1007/BFb0018999

Baader, F., Horrocks, I., & Sattler, U. (2008). Description Logics. In F. van Harmelen, V. Lifschitz, & B. Porter (Eds.), *Handbook of knowledge representation* (pp. 135–180). Elsevier. Retrieved from `download/2007/BaHS07a.pdf`

Baader, F., Lutz, C., & Brandt, S. (2008). Pushing the $\mathcal{EL}$ Envelope Further. In *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions, Washington, DC, USA, 1-2 April 2008.* Retrieved from `http://ceur-ws.org/Vol-496/owled2008dc_paper_3.pdf`

Baader, F., Lutz, C., & Suntisrivaraporn, B. (2005). Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice. In *Proceedings of the 2005 International Workshop on Methods for Modalities (M4M-05).*

Baader, F., Lutz, C., & Suntisrivaraporn, B. (2006). CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach & N. Shankar (Eds.), *Proceedings of the 3rd international joint conference on automated reasoning (IJCAR'06)* (Vol. 4130, pp. 287–291). Springer-Verlag.

Baader, F., & Nutt, W. (2003). Basic Description Logics. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, & P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation and Applications* (p. 43-95). New York, NY, USA: Cambridge University Press. Retrieved from `http://dl.acm.org/citation.cfm?id=885746.885749`

Baader, F., & Sattler, U. (2001). An overview of tableau algorithms for description logics. *Studia Logica*, *69*(1), 5–40. Retrieved from `http://dx.doi.org/10.1023/A:1013882326814` doi: 10.1023/A:1013882326814

Baader, F., & Sattler, U. (2003). Description logics with aggregates and concrete domains. *Information Systems*, *28*(8), 979–1004.

Bacchus, F., & Van Run, P. (1995). Dynamic variable ordering in csps. In *International conference on principles and practice of constraint programming* (pp. 258–275).

Bachmair, L., & Ganzinger, H. (1997). A theory of resolution.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., & Stein, L. A. (2004, February). *OWL Web Ontology Language Reference.* W3C Recommendation. Retrieved from `https://www.w3.org/TR/owl-ref/`

Becket, R. (2014). *Specification of FlatZinc-Version 1.6.* NICTA, Victoria Research Lab, Melbourne, Australia.

Bessiere, C. (2006). Constraint propagation. In *Handbook of constraint programming* (Vol. 2, p. 29 - 83). Elsevier.

Bessiere, C., & Régin, J.-C. (1996). MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *International conference on principles and practice of constraint programming* (pp. 61–75).

Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.). (2009). *Handbook of satisfiability* (Vol. 185). IOS Press.

Bijarbooneh, F. H., Pathak, A., Pearson, J., Issarny, V., & Jonsson, B. (2014). A Constraint Programming Approach for Managing End-to-end Requirements in Sensor Network Macroprogramming. In *SENSORNETS 2014 - Proceedings of the 3rd International Conference on Sensor Networks, Lisbon, Portugal, 7 - 9 January, 2014* (pp. 28–40). Retrieved from `http://dx.doi.org/10.5220/0004715200280040` doi: 10.5220/0004715200280040

Bobrow, D. G., & Raphael, B. (1974, September). New programming languages for artificial intelligence research. *ACM Comput. Surv.*, *6*(3), 153–174. Retrieved from `http://doi.acm.org.ezproxy.lib.monash.edu.au/10.1145/356631.356632` doi: 10.1145/356631.356632

Boussemart, F., Hemery, F., Lecoutre, C., & Sais, L. (2004). Boosting systematic search by weighting constraints. In *ECAI* (Vol. 16, p. 146).

Brandt, S. (2004). Polynomial Time Reasoning in a Description Logic with Existential Restrictions, GCI Axioms, and—What Else? In R. L. de Mantáras & L. Saitta (Eds.), *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004)* (pp. 298–302). IOS Press.

Buchheit, M., Donini, F. M., & Schaerf, A. (1993, December). Decidable reasoning in terminological knowledge representation systems. *J. Artif. Int. Res.*, *1*(1), 109–138. Retrieved from `http://dl.acm.org/citation.cfm?id=1618595.1618601`

Calvanese, D., Kharlamov, E., Nutt, W., & Thorne, C. (2008). Aggregate queries over ontologies. In *Proceedings of the 2nd international workshop on ontologies and information systems for the semantic web* (pp. 97–104).

Carroll, J., Herman, I., & Patel-Schneider, P. F. (2012, December). *OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition).* W3C Recommendation. Retrieved from `https://www.w3.org/TR/owl2-rdf-based-semantics/`

Chandrasekaran, B., Josephson, J. R., & Benjamins, R. V. (1999, January). What are Ontologies and why do we need them? *IEEE Intelligent Systems*, 20–26.

Chen, H., Perich, F., Finin, T. W., & Joshi, A. (2004). SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services, 22-25 August 2004, Cambridge, MA, USA* (pp. 258–267). Retrieved from `http://dx.doi.org/10.1109/MOBIQ.2004.1331732` doi: 10.1109/MOBIQ.2004.1331732

Chu, G. G. (2011). *Improving combinatorial optimization* (Unpublished doctoral dissertation). The University of Melbourne.

Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., & Stenico, C. (2010). Satisfiability Modulo the Theory of Costs: Foundations and Applications. In J. Esparza & R. Majumdar (Eds.), *Tools and algorithms for the construction and analysis of systems* (Vol. 6015, p. 99-113). Springer Berlin Heidelberg.

Cohen, D. A., Jeavons, P., Jefferson, C., Petrie, K. E., & Smith, B. M. (2006). Symmetry definitions for constraint satisfaction problems. *Constraints*, *11*(2-3), 115–137. Retrieved from `http://dx.doi.org/10.1007/s10601-006-8059-8` doi: 10.1007/s10601-006-8059-8

Crawford, J. M., Ginsberg, M. L., Luks, E. M., & Roy, A. (1996). Symmetry-Breaking Predicates for Search Problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996.* (pp. 148–159).

Cyganiak, R., Wood, D., & Lanthaler, M. (Eds.). (2014, February). *RDF 1.1 Concepts and Abstract Syntax.* W3C Recommendation. Retrieved from `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`

Davis, M., Logemann, G., & Loveland, D. W. (1962). A machine program for theorem-proving. *Commun. ACM*, *5*(7), 394–397. Retrieved from `http://doi.acm.org/10.1145/368273.368557` doi: 10.1145/368273.368557

Dawson, R. (2011). How significant is a boxplot outlier. *Journal of Statistics Education*, *19*(2), 1–12.

Dechter, R., & Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, *34*(1), 1–38.

de la Barra, C. L., Soto, R., Crawford, B., Allendes, C., Berendsen, H., & Monfroy, E. (2013). Modeling the portfolio selection problem with constraint programming. In *International conference on human-computer interaction* (pp. 645–649).

Donini, F. M. (2003). Complexity of reasoning. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, & P. F. Patel-Schneider (Eds.), *The Description Logic Handbook:*

Theory, Implementation and Applications (pp. 96–136). New York, NY, USA: Cambridge University Press. Retrieved from `http://dl.acm.org/citation.cfm?id=885746.885750`

Drummond, N., Horridge, M., Stevens, R., Wroe, C., & Sampaio, S. (2007). Pizza ontology. *The University of Manchester*, *2*.

Feder, T., & Hell, P. (2006). Full constraint satisfaction problems. *SIAM Journal on Computing*, *36*(1), 230–246.

Feydy, T., & Stuckey, P. J. (2009). Lazy clause generation reengineered. In *Proceedings of the 15th international conference on principles and practice of constraint programming* (pp. 352–366). Berlin, Heidelberg: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=1788994.1789026`

Flener, P., Frisch, A. M., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., & Walsh, T. (2002). Breaking Row and Column Symmetries in Matrix Models. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings* (pp. 462–476). Retrieved from `http://link.springer.de/link/service/series/0558/bibs/2470/24700462.htm`

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *Journal of the ACM (JACM)*, *29*(1), 24–32.

Freuder, E. C., & Mackworth, A. K. (2006). Constraint satisfaction: An emerging paradigm. In *Handbook of constraint programming* (pp. 13–27). Retrieved from `http://dx.doi.org/10.1016/S1574-6526(06)80006-4` doi: 10.1016/S1574-6526(06)80006-4

Frisch, A. M., Jefferson, C., & Miguel, I. (2003). Constraints for Breaking More Row and Column Symmetries. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings* (pp. 318–332). Retrieved from `http://dx.doi.org/10.1007/978-3-540-45193-8_22` doi: 10.1007/978-3-540-45193-8_22

Frost, D., Dechter, R., et al. (1995). Look-ahead value ordering for constraint satisfaction problems. In *IJCAI (1)* (pp. 572–578).

Fruhwirth, T., & Abdennadher, S. (2006). Principles of constraint systems and constraint solvers. *Archives of Control Sciences*, *16*(2), 131.

Gaschnig, J. G. (1979). *Performance measurement and analysis of certain search algorithms.* (Unpublished doctoral dissertation). Pittsburgh, PA, USA. (AAI7925014)

Geelen, P. A. (1992). Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the 10th european conference on artificial intelligence* (pp. 31–35). New York, NY, USA: John Wiley & Sons, Inc. Retrieved from `http://dl.acm.org/citation.cfm?id=145448.145491`

Gent, I. P., MacIntyre, E., Presser, P., Smith, B. M., & Walsh, T. (1996). An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *International conference on principles and practice of constraint programming* (pp. 179–193).

Gent, I. P., Petrie, K. E., & Puget, J. (2006). Symmetry in constraint programming. In *Handbook of constraint programming* (pp. 329–376). Retrieved from `http://dx.doi.org/10.1016/S1574-6526(06)80014-3` doi: 10.1016/S1574-6526(06)80014-3

Ginsberg, M. L., Frank, M., Halpin, M. P., & Torrance, M. C. (1990). Search Lessons Learned from Crossword Puzzles. In *AAAI* (Vol. 90, pp. 210–215).

Glimm, B., Horrocks, I., Motik, B., Stoilos, G., & Wang, Z. (2014). HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, *53*(3), 245–269.

Golbeck, J., Fragoso, G., Hartel, F., Hendler, J., Oberthaler, J., & Parsia, B. (2003). The national cancer institute's thesaurus and ontology. *Web Semantics: Science,* *Services and Agents on the World Wide Web*, *1*(1). Retrieved from `http://imap.websemanticsjournal.org/index.php/ps/article/view/27`

Golomb, S. W., & Baumert, L. D. (1965). Backtrack programming. *Journal of the ACM (JACM)*, *12*(4), 516–524.

Grau, B. C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P. F., & Sattler, U. (2008). OWL 2: The next step for OWL. *J. Web Sem.*, *6*(4), 309–322. Retrieved from `http://dx.doi.org/10.1016/j.websem.2008.05.001` doi: 10.1016/j.websem.2008.05.001

Grosof, B. N., Horrocks, I., Volz, R., & Decker, S. (2003). Description Logic Programs: Combining Logic Programs with Description Logic. In *Proceedings of the twelfth international world wide web conference, WWW 2003, budapest, hungary, may 20-24, 2003* (pp. 48–57). Retrieved from `http://doi.acm.org/10.1145/775152.775160` doi: 10.1145/775152.775160

Guns, T., Dries, A., Tack, G., Nijssen, S., & De Raedt, L. (2013). Miningzinc: A modeling language for constraint-based mining. In *Proceedings of the twenty-third international joint conference on artificial intelligence* (pp. 1365–1372).

Haarslev, V., Lutz, C., & Möller, R. (1999). A description logic with concrete domains and a role-forming predicate operator. *Journal of Logic and Computation*, *9*(3), 351–384.

Haarslev, V., Möller, R., & Wessel, M. (2001). The Description Logic $\mathcal{ALCNH}_{R^+}$ Extended with Concrete Domains: A Practically Motivated Approach. In *Proceedings of the first international joint conference on automated reasoning* (pp. 29–44). London, UK, UK: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=648237.753934`

Haarslev, V., Sebastiani, R., & Vescovi, M. (2011). Automated Reasoning in $\mathcal{ALCQ}$ via SMT. In N. Bjørner & V. Sofronie-Stokkermans (Eds.), *Automated Deduction – CADE-23* (Vol. 6803, p. 283-298). Springer Berlin Heidelberg.

Haralick, R. M., & Elliott, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, *14*(3), 263–313.

Heflin, J. (Ed.). (2004, February 10). *OWL Web Ontology Language Use Cases and Requirements.* W3C Recommendation. Retrieved from `http://www.w3.org/TR/webont-req/#onto-def` ([online] `http://www.w3.org/TR/webont-req/#onto-def`)

Hollunder, B. (1990). Hybrid Inferences in KL-ONE-Based Knowledge Representation Systems. In *Proceedings of the 14th German Workshop on Artificial Intelligence* (pp. 38–47). London, UK, UK: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=647610.733481`

Hollunder, B., & Baader, F. (1991). *Qualifying number restrictions in concept languages* (Tech. Rep.). Postfach 151141, 66041 Saarbrücken: Saarländische Universitäts- und Landesbibliothek. Retrieved from `http://scidok.sulb.uni-saarland.de/volltexte/2011/3562`

Hollunder, B., Nutt, W., & Schmidt-Schauß, M. (1990). Subsumption algorithms for concept description languages. In *ECAI* (pp. 348–353).

Hoos, H. H. (2011). *Satlib - benchmark problems.* Retrieved March, 2014, from `http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`

Horrocks, I. (1998). Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)* (pp. 636–647). Retrieved from `download/1998/kr98.pdf`

Horrocks, I. (2003). Implementation and optimization techniques. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, & P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation and Applications* (pp. 306–346). New

York, NY, USA: Cambridge University Press. Retrieved from http://dl.acm.org/citation.cfm?id=885746.885756

Horrocks, I., Kutz, O., & Sattler, U. (2006). The Even More Irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)* (pp. 57–67). AAAI Press. Retrieved from download/2006/HoKS06a.pdf

Horrocks, I., Parsia, B., & Sattler, U. (2012, December). *OWL 2 Web Ontology Language Direct Semantics (Second Edition).* W3C Recommendation. Retrieved from https://www.w3.org/TR/owl2-direct-semantics/

Horrocks, I., & Patel-Schneider, P. F. (2004). Reducing OWL entailment to description logic satisfiability. *J. Web Sem.*, *1*(4), 345–357. Retrieved from http://dx.doi.org/10.1016/j.websem.2004.06.003 doi: 10.1016/j.websem.2004.06.003

Horrocks, I., & Patel-Schneider, P. F. (1998). DL systems comparison. In *Proc. of the 1998 Description Logic Workshop (DL'98)* (Vol. 11, pp. 55–57).

Horrocks, I., & Patel-Schneider, P. F. (2003). Reducing owl entailment to description logic satisfiability. In *International semantic web conference* (pp. 17–29).

Horrocks, I., Patel-Schneider, P. F., & van Harmelen, F. (2003). From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language. *J. of Web Semantics*, *1*(1), 7–26. Retrieved from download/2003/HoPH03a.pdf

Horrocks, I., & Sattler, U. (2001). Ontology Reasoning in the $\mathcal{SHOQ}(\mathcal{D})$ Description Logic. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001* (pp. 199–204).

Horrocks, I., & Sattler, U. (2005). A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005* (pp. 448–453). Retrieved from http://ijcai.org/Proceedings/05/Papers/0759.pdf

Horrocks, I., & Sattler, U. (2007). A tableau decision procedure for $\mathcal{SHOIQ}$. *J. of Automated Reasoning*, *39*(3), 249–276. Retrieved from download/2007/HoSa07a.pdf doi: 10.1007/s10817-007-9079-9

Horrocks, I., Sattler, U., & Tobies, S. (1999). Practical Reasoning for Expressive Description Logics. In H. Ganzinger, D. Mcallester, & A. Voronkov (Eds.), *Proceedings of the 6th international conference on logic for programming and automated reasoning (LPAR'99)* (pp. 161–180). Springer-Verlag. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.5145

Hustadt, U., de Nivelle, H., & Schmidt, R. A. (2000). Resolution-Based Methods for Modal Logics. *Logic Journal of the IGPL*, *8*(3), 265–292. Retrieved from https://doi.org/10.1093/jigpal/8.3.265 doi: 10.1093/jigpal/8.3.265

Jaulin, L. (2016). Range-only slam with indistinguishable landmarks; a constraint programming approach. *Constraints*, *21*(4), 557–576. Retrieved from http://dx.doi.org/10.1007/s10601-015-9231-9 doi: 10.1007/s10601-015-9231-9

Kamp, G., & Wache, H. (1996). *CTL : a description logic with expressive concrete domains* (Tech. Rep.). Postfach 151141, 66041 Saarbrücken: Saarländische Universitäts- und Landesbibliothek. Retrieved from http://scidok.sulb.uni-saarland.de/volltexte/2011/3903

Kang, Y.-B., Krishnaswamy, S., & Li, Y.-F. (2015). R2O2: an efficient ranking-based reasoner for OWL ontologies. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I* (pp. 322–338). Retrieved from http://dx.doi.org/10.1007/978-3-319-25007-6_19 doi: 10.1007/978-3-319-25007-6_19

Kang, Y.-B., Li, Y.-F., & Krishnaswamy, S. (2012). Predicting reasoning performance using ontology metrics. In *The Semantic Web–ISWC 2012* (pp. 198–214). Springer.

Kang, Y.-B., Pan, J. Z., Krishnaswamy, S., Sawangphol, W., & Li, Y.-F. (2014). How long will it take? accurate prediction of ontology reasoning performance. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.* (pp. 80–86). Retrieved from `http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8305`

Kazakov, Y. (2008). $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008* (pp. 274–284). Retrieved from `http://www.aaai.org/Library/KR/2008/kr08-027.php`

Kazakov, Y. (2009). Consequence-driven reasoning for horn $\mathcal{SHIQ}$ ontologies. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009* (pp. 2040–2045). Retrieved from `http://ijcai.org/Proceedings/09/Papers/336.pdf`

Kazakov, Y., & Klinov, P. (2014a). Bridging the Gap between Tableau and Consequence-Based Reasoning. In *Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014.* (pp. 579–590). Retrieved from `http://ceur-ws.org/Vol-1193/paper_10.pdf`

Kazakov, Y., & Klinov, P. (2014b). Goal-Directed Tracing of Inferences in $\mathcal{EL}$ Ontologies. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II* (pp. 196–211). Retrieved from `http://dx.doi.org/10.1007/978-3-319-11915-1_13` doi: 10.1007/978-3-319-11915-1_13

Kazakov, Y., Kroetzsch, M., & Simancik, F. (2012). Practical Reasoning with Nominals in the EL Family of Description Logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012.* Retrieved from `http://www.aaai.org/ocs/index.php/KR/KR12/paper/view/4540`

Kazakov, Y., Krötzsch, M., & Simancík, F. (2011). Concurrent classification of $\mathcal{EL}$ ontologies. In *Proceedings of the 10th international conference on the semantic web - volume part i* (pp. 305–320). Berlin, Heidelberg: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=2063016.2063037`

Kazakov, Y., Krötzsch, M., & Simancik, F. (2011). Unchain My $\mathcal{EL}$ Reasoner. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011), Barcelona, Spain, July 13-16, 2011.* Retrieved from `http://ceur-ws.org/Vol-745/paper_54.pdf`

Kazakov, Y., Krötzsch, M., & Simancik, F. (2014). The incredible ELK - from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. *J. Autom. Reasoning*, *53*(1), 1–61. Retrieved from `http://dx.doi.org/10.1007/s10817-013-9296-3` doi: 10.1007/s10817-013-9296-3

Kharlamov, E., Kotidis, Y., Mailis, T., Neuenstadt, C., Nikolaou, C., Özçep, Ö. L., ... Möller, R. (2016). Towards analytics aware ontology based access to static and streaming data (extended version). *CoRR*, *abs/1607.05351.* Retrieved from `http://arxiv.org/abs/1607.05351`

Kostov, B., & Křemen, P. (2013). Count aggregation in semantic queries. In *Proceedings of the 9th International Conference on Scalable Semantic Web Knowledge Base Systems-Volume 1046* (pp. 1–16).

Kostylev, E. V., & Reutter, J. L. (2013). Answering Counting Aggregate Queries over Ontologies of the DL-Lite Family. In *AAAI.*

Krötzsch, M., Simancik, F., & Horrocks, I. (2012). A description logic primer. *CoRR*, *abs/1201.4089.*

Kullmann, M., de Beuvron, F. d. B., & Rousselot, F. (2000). A description logic model for reacting in a dynamic environment. In *Proceedings of the 2000 International Workshop in Description Logics (DL2000), number 33 in CEUR-WS.*

Lecoutre, C., Boussemart, F., & Hemery, F. (2004). Backjump-based techniques versus conflict-directed heuristics. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on* (pp. 549–557).

Lécué, F. (2012). Diagnosing Changes in An Ontology Stream: A DL Reasoning Approach. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* Retrieved from `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4988`

Lécué, F., Tallevi-Diotallevi, S., Hayes, J., Tucker, R., Bicer, V., Sbodio, M. L., & Tommasi, P. (2014). Star-city: semantic traffic analytics and reasoning for city. In *Proceedings of the 19th international conference on Intelligent User Interfaces* (pp. 179–188).

Lei, Y., Uren, V. S., & Motta, E. (2006). SemSearch: A Search Engine for the Semantic Web. In *Managing Knowledge in a World of Networks, 15th International Conference, EKAW 2006, Podebrady, Czech Republic, October 2-6, 2006, Proceedings* (pp. 238–245). Retrieved from `http://dx.doi.org/10.1007/11891451_22` doi: 10.1007/11891451_22

Lu, Q., & Tosic, V. (2010). Minimasc+ minizinc: An autonomic business-driven decision making middleware for adaptation of web service compositions. In *Ubiquitous intelligence & computing and 7th international conference on autonomic & trusted computing (uic/atc), 2010 7th international conference on* (pp. 474–477).

Luks, E. M., & Roy, A. (2004). The complexity of symmetry-breaking formulas. *Ann. Math. Artif. Intell.*, *41*(1), 19–45. Retrieved from `http://dx.doi.org/10.1023/B:AMAI.0000018578.92398.10` doi: 10.1023/B:AMAI.0000018578.92398.10

Lutz, C. (2002). PSPACE reasoning with the description logic $\mathcal{ALCF}(\mathcal{D})$. *Logic Journal of IGPL*, *10*(5), 535–568.

Lutz, C., Areces, C., Horrocks, I., & Sattler, U. (2003). Keys, nominals, and concrete domains. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003* (pp. 349–354).

Lutz, C., & Milicic, M. (2004). Description logics with concrete domains and functional dependencies. In *ECAI* (Vol. 16, p. 378).

Lutz, C., & Milicic, M. (2007). A Tableau Algorithm for Description Logics with Concrete Domains and General TBoxes. *J. Autom. Reasoning*, *38*(1-3), 227–259. Retrieved from `http://dx.doi.org/10.1007/s10817-006-9049-7` doi: 10.1007/s10817-006-9049-7

Mackworth, A. K. (1977a). Consistency in networks of relations. *Artif. Intell.*, *8*(1), 99–118. Retrieved from `http://dx.doi.org/10.1016/0004-3702(77)90007-8` doi: 10.1016/0004-3702(77)90007-8

Mackworth, A. K. (1977b). On Reading Sketch Maps. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, August 22-25, 1977* (pp. 598–606). Retrieved from `http://ijcai.org/Proceedings/77-2/Papers/006.pdf`

McBride, S., Lawley, M., Leroux, H., & Gibson, S. (2012). Using Australian Medicines Terminology (AMT) and SNOMED CT-AU to better support clinical research. *Studies in health technology and informatics*, *178*, 144–149.

McGuinness, D. L., & van Harmelen, F. (Eds.). (2004, February). *OWL Web Ontology Language Overview.* W3C Recommendation. Retrieved from `https://www.w3.org/TR/owl-features/`

Meissner, A. (2011). The $\mathcal{ALCN}$ description logic concept satisfiability as a sat problem. In R. Katarzyniak, T.-F. Chiu, C.-F. Hong, & N. Nguyen (Eds.), *Semantic methods* *for knowledge management and communication* (Vol. 381, p. 253-263). Springer Berlin Heidelberg.

Mendez, J., & Suntisrivaraporn, B. (2009). Reintroducing CEL as an OWL 2 EL reasoner. In B. C. Grau, I. Horrocks, B. Motik, & U. Sattler (Eds.), *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009* (Vol. 477). CEUR-WS.org.

Métivier, J., Loudni, S., & Charnois, T. (2013). A constraint programming approach for mining sequential patterns in a sequence database. *CoRR*, *abs/1311.6907*. Retrieved from `http://arxiv.org/abs/1311.6907`

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001* (pp. 530–535). Retrieved from `http://doi.acm.org/10.1145/378239.379017` doi: 10.1145/378239.379017

Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., & Lutz, C. (Eds.). (2012, December 11). *OWL 2 Web Ontology Language Profiles.* W3C Recommendation. Retrieved from `http://www.w3.org/TR/owl-profiles/` ([online] `http://www.w3.org/TR/owl-profiles/`)

Motik, B., Shearer, R., & Horrocks, I. (2007a). A Hypertableau Calculus for $\mathcal{SHIQ}$. In *Proc. of the 2007 Description Logic Workshop (DL 2007)* (Vol. 250). Retrieved from `download/2007/MoSH07b.pdf`

Motik, B., Shearer, R., & Horrocks, I. (2007b). Optimized Reasoning in Description Logics using Hypertableaux. In *Proc. of the 21st Int. Conf. on Automated Deduction (CADE-21)* (Vol. 4603, pp. 67–83). Springer. Retrieved from `download/2007/MoSH07a.pdf`

Motik, B., Shearer, R., & Horrocks, I. (2009). Hypertableau reasoning for description logics. *J. of Artificial Intelligence Research*, *36*, 165–228. Retrieved from `download/2009/MoSH09a.pdf`

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). MiniZinc: Towards a standard CP modelling language. In C. Bessière (Ed.), *Principles and Practice of Constraint Programming – CP 2007* (Vol. 4741, p. 529-543). Springer Berlin Heidelberg.

Ohrimenko, O., Stuckey, P., & Codish, M. (2007). Propagation = Lazy Clause Generation. In C. Bessière (Ed.), *Principles and Practice of Constraint Programming – CP 2007* (Vol. 4741, p. 544-558). Springer Berlin Heidelberg. Retrieved from `http://dx.doi.org/10.1007/978-3-540-74970-7_39` doi: 10.1007/978-3-540-74970-7_39

Osumi-Sutherland, D., Reeve, S., Mungall, C. J., Neuhaus, F., Ruttenberg, A., Jefferis, G. S. X. E., & Armstrong, J. D. (2012). A strategy for building neuroanatomy ontologies. *Bioinformatics*, *28*(9), 1262–1269. Retrieved from `http://dx.doi.org/10.1093/bioinformatics/bts113` doi: 10.1093/bioinformatics/bts113

Pan, J. Z. (2007). A flexible ontology reasoning architecture for the semantic web. *Knowledge and Data Engineering, IEEE Transactions on*, *19*(2), 246–260.

Patel-Schneider, P. F., Hayes, P., & Horrocks, I. (Eds.). (2004, February). *OWL Web Ontology Language Semantics and Abstract Syntax.* W3C Recommendation. Retrieved from `https://www.w3.org/TR/owl-semantics/`

Petrie, K. E., & Smith, B. M. (2003). Symmetry Breaking in Graceful Graphs. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings* (pp. 930–934). Retrieved from `http://dx.doi.org/10.1007/978-3-540-45193-8_81` doi: 10.1007/978-3-540-45193-8_81

Puget, J. (1998). A Fast Algorithm for the Bound Consistency of alldiff Constraints. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and*

Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA. (pp. 359–366). Retrieved from `http://www.aaai.org/Library/AAAI/1998/aaai98-051.php`

Puget, J.-F. (1993). On the satisfiability of symmetrical constrained satisfaction problems. In *Proceedings of the 7th international symposium on methodologies for intelligent systems* (pp. 350–361). London, UK, UK: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=646354.688961`

Puget, J.-F. (2005). Symmetry breaking revisited. *Constraints*, *10*(1), 23–46.

Rector, A. L., Rogers, J. E., & Pole, P. A. (1996, January). The GALEN High Level Ontology. *Proceedings MIE 96*, 174–178.

Rossi, F., Beek, P. v., & Walsh, T. (2006). *Handbook of Constraint Programming.* Elsevier.

Rossi, F., van Beek, P., & Walsh, T. (2006). Introduction. In P. v. B. Francesca Rossi & T. Walsh (Eds.), *Handbook of constraint programming* (Vol. 2, p. 3 - 12). Elsevier. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1574652606800052` doi: http://dx.doi.org/10.1016/S1574-6526(06)80005-2

Samwald, M. (2013). Genomic CDS: an example of a complex ontology for pharmacogenetics and clinical decision support. In *2nd OWL Reasoner Evaluation Workshop (ORE 2013)* (p. 128).

Schild, K. (1991). A Correspondence Theory for Terminological Logics: Preliminary Report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991* (pp. 466–471). Retrieved from `http://ijcai.org/Proceedings/91-1/Papers/072.pdf`

Schmidt, R. A. (2006). Developing Modal Tableaux and Resolution Methods via First-Order Resolution. In *Advances in Modal Logic 6, papers from the sixth conference on "Advances in Modal Logic," held in Noosa, Queensland, Australia, on 25-28 September 2006* (pp. 1–26). Retrieved from `http://www.aiml.net/volumes/volume6/Schmidt.ps`

Schmidt, R. A., & Hustadt, U. (2013). First-Order Resolution Methods for Modal Logics. In *Programming Logics - Essays in Memory of Harald Ganzinger* (pp. 345–391). Retrieved from `https://doi.org/10.1007/978-3-642-37651-1_15` doi: 10.1007/978-3-642-37651-1_15

Schmidt-Schauß, M., & Smolka, G. (1991). Attributive concept descriptions with complements,. *Artificial Intelligence*, *48*, 1–26.

Shearer, R., Motik, B., & Horrocks, I. (2008, October 26–27). HermiT: A Highly-Efficient OWL Reasoner. In A. Ruttenberg, U. Sattler, & C. Dolbear (Eds.), *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU).* Karlsruhe, Germany.

Sidhu, A. S., Dillon, T. S., Chang, E., & Sidhu, B. S. (2005). Protein Ontology Development using OWL. In *Proceedings of the OWLED*05 Workshop on OWL: Experiences and Directions, Galway, Ireland, November 11-12, 2005.* Retrieved from `http://ceur-ws.org/Vol-188/sub35.pdf`

Simancik, F., Kazakov, Y., & Horrocks, I. (2011). Consequence-Based Reasoning beyond Horn Ontologies. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011* (pp. 1093–1098). Retrieved from `http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-187` doi: 10.5591/978-1-57735-516-8/IJCAI11-187

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, *5*(2), 51 - 53. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1570826807000169` (Software Engineering and the Semantic Web) doi: http://dx.doi.org/10.1016/j.websem.2007.03.004

Smith, B. M. (1996). Succeed-first or fail-first: A case study in variable and value ordering. In *Proceedings of The Third International Conference on the Practical Applications of Constraint Technology (PACT'97). Practical Applications Company, Blackpool, UK* (pp. 321–330). Citeseer.

Stearns, M. Q., Price, C., Spackman, K. A., & Wang, A. Y. (2001). SNOMED clinical terms: overview of the development process and project status. In *Proceedings of the AMIA Symposium* (p. 662).

Steigmiller, A., Liebig, T., & Glimm, B. (2012). Extended caching, backjumping and merging for expressive description logics. In B. Gramlich, D. Miller, & U. Sattler (Eds.), *Automated reasoning* (Vol. 7364, p. 514-529). Springer Berlin Heidelberg.

Steigmiller, A., Liebig, T., & Glimm, B. (2014). Konclude: System description. *Journal of Web Semantics (JWS)*, *27*, 78–85.

Stuckey, P. J., de la Banda, M. J. G., Maher, M. J., Marriott, K., Slaney, J. K., Somogyi, Z., ... Walsh, T. (2005). The G12 Project: Mapping Solver Independent Models to Efficient Solutions. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings* (pp. 13–16). Retrieved from `http://dx.doi.org/10.1007/11564751_4` doi: 10.1007/11564751_4

Suntisrivaraporn, B. (2009). *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies* (Doctoral dissertation, Dresden University of Technology, Germany). Retrieved from `http://hsss.slub-dresden.de/deds-access/hsss.urlmapping.MappingServlet?id=1233830966436-5928`

Tack, G. (2009). *Constraint Propagation - Models, Techniques, Implementation* (Doctoral dissertation, Saarland University, Germany). Retrieved from `http://www.gecode.org/paper.html?id=Tack:PhD:2009`

Tamma, V. A. M., Phelps, S., Dickinson, I., & Wooldridge, M. (2005). Ontologies for supporting negotiation in e-commerce. *Eng. Appl. of AI*, *18*(2), 223–236. Retrieved from `http://dx.doi.org/10.1016/j.engappai.2004.11.011` doi: 10.1016/j.engappai.2004.11.011

Thomas, E., Pan, J. Z., & Ren, Y. (2010). TrOWL: Tractable OWL 2 reasoning infrastructure. In *The Semantic Web: Research and Applications* (pp. 431–435). Springer.

Tobies, S. (2001). *Complexity results and practical algorithms for logics in knowledge representation* (Doctoral dissertation, RWTH Aachen University, Germany). Retrieved from `http://sylvester.bth.rwth-aachen.de/dissertationen/2001/082/01_082.pdf`

Tsarkov, D., & Horrocks, I. (2006). FaCT++ Description Logic Reasoner: System Description. In U. Furbach & N. Shankar (Eds.), *Automated reasoning* (Vol. 4130, pp. 292–297). Berlin, Heidelberg: Springer Berlin Heidelberg.

Turhan, A.-Y. (2010). Reasoning and explanation in $\mathcal{EL}$ and in expressive description logics. In *Proceedings of the 6th international conference on semantic technologies for software engineering* (pp. 1–27). Berlin, Heidelberg: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=1886135.1886136`

van Beek, P. (2006). Chapter 4 - backtracking search algorithms. In P. v. B. Francesca Rossi & T. Walsh (Eds.), *Handbook of constraint programming* (Vol. 2, p. 85 - 134). Elsevier. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1574652606800088` doi: http://dx.doi.org/10.1016/S1574-6526(06)80008-8

Walsh, T. (2006). General Symmetry Breaking Constraints. In *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings* (pp. 650–664). Retrieved from `http://dx.doi.org/10.1007/11889205_46` doi: 10.1007/11889205_46

Wang, X., Zhang, D., Gu, T., & Pung, H. K. (2004). Ontology Based Context Modeling and Reasoning using OWL. In *2nd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2004 Workshops), 14-17 March 2004, Orlando, FL, USA* (pp. 18–22). Retrieved from `http://dx.doi.org/10.1109/PERCOMW.2004.1276898` doi: 10.1109/PERCOMW.2004.1276898

Zabih, R. (1990). Some Applications of Graph Bandwidth to Constraint Satisfaction Problems. In *AAAI* (pp. 46–51).

Zhang, L., Madigan, C. F., Moskewicz, M. H., & Malik, S. (2001). Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design* (pp. 279–285).

Zuo, M., & Haarslev, V. (2013). Intelligent Tableau Algorithm for DL Reasoning. In *Automated Reasoning with Analytic Tableaux and Related Methods* (pp. 273–287). Springer.