TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science International Center for Computational Logic

Institute of Theoretical Computer Science, Chair of Automata Theory

A Polynomial Approach for finding Most Specific Concept w.r.t General $\mathcal{EL}$-TBoxes

Mohamed Ali Abdelmajid Labib Nadeem

Master Thesis

TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science International Center for Computational Logic

Institute of Theoretical Computer Science, Chair of Automata Theory

A Polynomial Approach for finding Most Specific Concept w.r.t General $\mathcal{EL}$-TBoxes

Mohamed Ali Abdelmajid Labib Nadeem

Born on: 10th October 1994 in Egypt
Course: Computational Logic
Discipline: Automata Theory
Matriculation number: 4870360
Matriculation year: 2019

Master Thesis

to achieve the academic degree

Master of Science (M.Sc.)

Supervisor
Dr. Patrick Koopmann

Supervising professor
Prof. Dr. Sebastian Rudolph

Submitted on: 2nd September 2022

TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science International Center for Computational Logic

Institute of Theoretical Computer Science, Chair of Automata Theory

Task for the preparation of a Master Thesis

Course:	Computational Logic
Discipline:	Automata Theory
Name:	Mohamed Ali Abdelmajid Labib Nadeem
Matriculation number:	4870360
Matriculation year:	2019
Title:	A Polynomial Approach for finding Most Specific Concept w.r.t General $\mathcal{EL}$-TBoxes

Objectives of work

The task to be solved in the Master's thesis is to implement and optimize a polynomial-time procedure that can test for the existence of the most specific concept and compute it in case it exists. Moreover, we provide a proof showing the correctness of this procedure. Finally, providing experimental results to show the compare it with the previous approach.

Focus of work

- Defining a polynomial-time procedure to check the existence of msc.
- Proving correctness and completeness of the defined procedure.
- Providing an implementation for the defined procedure.
- Providing an evaluation to compare the new approach with the previous approach.

Supervisor:	Dr. Patrick Koopmann
Issued on:	28.04.2022
Due date for submission:	06.10.2022

Prof. Dr. Sebastian Rudolph
Supervising professor

Acknowledgements

First, I would like to express my deepest gratitude and appreciation to my supervisor Dr Patrick Koopmann, whose help stimulating suggestions and encouragement helped me in all the research for and writing of this thesis. Through the collaboration in my Master's thesis, he taught me how to do scientific research. His guidance carried me through all stages of writing my thesis.

In addition, I want to thank Dr Johannes K. Fichte from the Vienna University of Technology. He is the one who motivated me by giving me comments and suggestions to improve my writing skills. I want to say that without working with him during my Master's project, I would not have continued in the field of Computational logic.

I would like also to thank Professor Sebastian Rudolph and Dr Adrian Nuradiansyah for their effort in reviewing my thesis.

I want to thank the secretaries and administrators, Mrs Romy Thieme and Mrs Sylvia Wünsch, for their help during my study.

I also thank my friends and colleagues, Islam Hamada and Piotr Gorczyca for their help in my study in several Courses throughout my Master's study.

I would also like to thank my family and friends, who have always supported and given me sound advice in my studies and life.

Finally, I wish to express special thanks to my mother, the only person in the world who believed in me and supported me throughout my life.

Abstract

Description logics (DLs) [Baa99; BHS04; Baa+17a; Tur13; KH17] are a family of formal knowledge representation languages used in artificial intelligence to describe and reason about the concepts of an application domain. In particular, they are providing formalization for ontologies and the semantic web. The most specific concept (msc) [BKM98] is an inference task that can support the bottom-up construction of knowledge bases in description logics. The most specific concept of an individual is the least concept that has this individual as an instance.

In description logics which contain existential restrictions, the most specific concept does not always exist in the case of acyclic ABoxes. However, the latest results [ZT13] show that the existence of the msc w.r.t. an individual can be decided in polynomial time. Also, the role-depth of these most specific generalizations is polynomially bounded by the size of the input, which yields a decision procedure for the existence problem. The polynomial bound can be used to compute the msc if it exists. Otherwise, the computed concept can still serve as an approximation[PT11]. However, computing the msc could take at least exponential time if the msc is exponentially large. Also, the previous approach does not make it clear how to compute the most specific concept in practice.

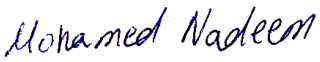
First, we revisit the previous approach for constructing the msc of an individual w.r.t. a general $\mathcal{EL}$-TBox. We present a new method for tree unravelling of an interpretation and introduce a characteristic concept w.r.t. the least tree unravelling. Moreover, we provide a new approach and an algorithm to decide the existence of the msc in polynomial time without relying on computing the concept. Then, one can compute the actual msc in exponential time. Finally, we provide an experimental evaluation to state that the concept constructed from the new approach has a smaller bound than the bound of the concept constructed from the previous approach.

Keywords— description logic, ontologies, most specific concept, polynomial time algorithms

Statement of authorship

I hereby certify that I have authored this document entitled *A Polynomial Approach for finding Most Specific Concept w.r.t General $\mathcal{EL}$-TBoxes* independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Dresden, 2nd September 2022



Mohamed Ali Abdelmajid Labib Nadeem

Restriction note

This document entitled *A Polynomial Approach for finding Most Specific Concept w.r.t General $\mathcal{EL}$-TBoxes* contains confidential data. Publications, duplications and inspections—even in part—are prohibited without explicit permission, as well as publications about the content of this document. It may only be made accessible to the supervisor at Technische Universität Dresden, the reviewers and also the members of the examination board.

Contents

List of Figures

List of Tables

1 Introduction

In philosophy, an ontology [Hof21] is the philosophical study of being in general, where concepts such as existence, being, becoming, and reality are studied. It includes the questions of how entities are grouped into basic categories and which of these entities exist on the most fundamental level. In computer science, an ontology [SS09] represents an abstract, simplified view of the relevant entities (objects, concepts, and relations) that exist in the domain of interest.

Ontology engineering is a field that studies methods for building an ontology. It aims to describe the knowledge of software applications, enterprises and business procedures for a particular domain. Supporting such applications require that the knowledge is presented precisely, so that it can be processed by *automated tools*. Ontology languages, such as OWL [Gra+08] is used to overcome this problem by defining a formal syntax and semantics to describe the knowledge of a particular domain.

Description Logics (DLs) are logic-based knowledge representation languages [Baa+17b] which are used to provide the formal foundation of the ontology language OWL for application domains such as the Semantic Web, biology and medicine, and engineering domains [HSG15]. It separates domain knowledge into a *terminological* part (called TBox) and an *assertional* part (called ABox), while the combination of both TBox and ABox is called a knowledge base (KB). The TBox describes the relations between concepts of an application domain, and the ABox describes the individual objects of an application domain and their relations. Concepts are built using concept and role names, and individual objects are built using concept and role assertions.

DLs are less expressive than first-order logic, but the core reasoning problems like the subsumption and the instance problem are usually decidable. Therefore, studying those problems is highly important in scientific and industrial fields. In applications of DL systems, it turned out that building and maintaining large DL knowledge bases can be facilitated by procedures for other, non-standard inference problems [BK06]. The non-standard inferences were introduced to maintain and support building large DL knowledge bases. An example of the non-standard inferences is generating a new concept from an individual (Also called the most specific concept of an individual).

Computing the most specific concept [Baa03a] is mainly used in the bottom-up construction of the knowledge base. This methodology allows the user to give an example of an individual belonging to the concept to be defined instead of defining the relevant concepts of an application domain from scratch. Thus, the knowledge engineer can use the computed concept as a starting point for defining the concept.

Moreover, since large ontologies might contain errors [Baa+21] often detected when DL reasoners compute unwanted consequences, it has gained interest in detecting errors in the data such that the unwanted consequences no longer follow, while keeping the other

consequences are preserved. An optimal repair is the repair with the least number of other consequences are removed. Most of the previous approaches of the ontology repairing rely on the syntactic nature of the ontology without relying on the semantic nature. The recent work [Baa+22] was addressed to compute the optimal repair of ABoxes, where the unwanted consequences are described as concept assertions. It also shows that the existence of such a type of repair can be decided by computing the most specific concept.

Unfortunately, the most specific concept does not always exist in $\mathcal{EL}$. It was investigated before by [Baa03a] in the presence of a cyclic description logic $\mathcal{EL}$-TBox (which allows for conjunctions, existential restrictions and the top concept), where the bottom-up construction of the knowledge base was used.

Example 1. consider the following knowledge base [KM02]:

$$\mathcal{K}_1 := (\emptyset, \mathcal{A}_1), \textit{ with } \mathcal{A}_1 := \{r(a,a), C(a)\}$$

The msc of the individual a does not exist, as the TBox $\mathcal{T}_1 = \emptyset$ is empty, while the ABox $\mathcal{A}_1$ is cyclic [Baa03a], due to the fact there is no least concept such that the individual a is an instance. The reason is that the concept C has more specific concept $\exists r.C$, and $\exists r.C$ has more specific concept $\exists\exists r.C$ and so on, while this cycle in $\mathcal{A}_1$ is not covered by any cycle in $\mathcal{T}_1$. However, consider the modified knowledge base:

$$\mathcal{K}_2 := (\mathcal{T}_2, \mathcal{A}_2), \textit{ with } \mathcal{A}_2 := \{r(a,a), C(a)\}, \textit{ and } \mathcal{T}_2 := \{C \sqsubseteq \exists r.C\}$$

Since the cycle in $\mathcal{A}_2$ is covered by $\mathcal{T}_2$, we obtain that C is the msc of a.

The first approach to decide the existence of the most specific concept [ZT13] shows that deciding the existence of the msc w.r.t. an individual can be done in polynomial time, where the construction of the generalization concept is computed up to a given k, a bound on the maximal nestings of quantifiers. However, this approach rely on computing the concept to decide the existence of the msc. Therefore, in practice it is still necessary to provide an efficient algorithm to decide the existence of the msc without relying on computing the concept.

Through the thesis, we provide an algorithm for deciding the most specific concept and prove its correctness and completeness. We show that the algorithm decides the existence of the msc in a polynomial run-time. Also, we provide an experimental evaluation to state the difference between the previous and new approach in terms of the bounded role-depth of the concept.

1.1 Thesis Structure

We begin with the preliminaries for the notation and terminology used throughout the thesis in Chapter 2. Then we start with the first part, which is about the first attempt to decide the existence of the most specific concept in Chapter 3. In Chapter 4, we introduce the least tree unravelling as a new approach for tree unravelling of an interpretation and show the difference between the new approach for tree unravelling and the previous one. Furthermore, we continue with the least tree unravelling of an interpretation and employ it for deciding the existence of the most specific concept. At the same time, we introduce a representation of interpretations as a graph and define a simulation between graphs.

Chapter 5 constitutes the final part of this thesis, in which we present the implementation. In the second part of this chapter, we prove that the algorithm runs in polynomial time. We also prove the correctness and completeness of the introduced algorithm.

Finally, in Chapter 6 we show experimental results to show the difference between the newly introduced approach and the previous approach in terms of the size of the bounded role-depth of the concept and the time required to decide the existence of the msc.

To lead the future work, In the final Chapter 7, we summarize our findings and point to some future work.

2 Preliminaries

This chapter introduces the general notation and terminology we use throughout this thesis. It is based on [Baa+17b] with some extensions. Note that these preliminaries are for description logics in general. In contrast, the background of more specific topics (e.g. the tree unravelling and the k-characteristic concept) will be given later when needed.

The chapter starts with an introduction of the description logic $\mathcal{EL}$. Then it introduces the syntax and semantics of the DL $\mathcal{EL}$ and continues with the definition of canonical models and simulation relations between two interpretations. In the end, it introduces the formal definition of the most specific concept of an individual.

2.1 The Description Logic $\mathcal{EL}$

We start with defining the syntax of $\mathcal{EL}$ and the knowledge base (KB). Then we define the semantics of $\mathcal{EL}$. A knowledge base is defined with the help of a set of constructors, starting with a set N_C of *concept names*, a set N_R of *role names* and a set N_I of *individual names*. The sets N_C, N_R and N_I are pairwise disjoint and countable infinite.

Definition 2.1.1 ($\mathcal{EL}$-concept)**.** Let $A \in N_C$ and $r \in N_R$, $\mathcal{EL}$-concepts C are then built according to the following syntax rule:

$$C ::= \top \mid A \mid C \sqcap D \mid \exists r.C$$

where $\top$ is the top concept, and D is an $\mathcal{EL}$-concept.

To be able measure the complexity of concepts, we need a notion of role-depth of concepts.

Definition 2.1.2 (role-depth)**.** Let C be a $\mathcal{EL}$-concept. The *role – depth* of the concept C ($rd(C)$) is inductively defined as follows:

- $rd(A) = rd(T) = 0$, for all $A \in C$.
- $rd(C \sqcap D) = max\{rd(C), rd(D)\}$, for all concepts C, D.
- $rd(\exists r.C) = 1 + rd(C)$, for all $r \in N_R$ and concepts C.

To illustrate this, consider the following example.

Example 2. Consider a concept $C := \exists r.(\exists r.(\exists s.A \sqcap \exists r.\exists s.T))$. Then $rd(C) = 3$.

Definition 2.1.3 (General concept inclusion)**.** Let C, D be $\mathcal{EL}$-concepts. A *general concept inclusion* (GCI) is an expression of the form $C \sqsubseteq D$. $C \equiv D$ is called a *concept definition* and it is an abbreviation for $C \sqsubseteq D$, $D \sqsubseteq C$.

A terminological TBox $\mathcal{T}$ is a finite set of concept definitions. A general TBox $\mathcal{T}$ is a finite set of GCIs.

Definition 2.1.4 (Acyclic TBox)**.** ([Baa+17a]) Let $\mathcal{T}$ be a TBox. We call $\mathcal{T}$ is an acyclic TBox if one of the following hold:

- There is no concept name in $\mathcal{T}$ that uses itself, and
- no concept name occurs more than once on the left-hand side of a concept definition in $\mathcal{T}$.

Otherwise, $\mathcal{T}$ is cyclic.

Definition 2.1.5 (Assertion)**.** Let $a,\ b \in N_I$, $r \in N_R$ and $C \in N_C$, then:

- $C(a)$ is called a *concept assertion*.
- $r(a, b)$ is called *role assertion*. a is called r-predecessor of b, and b is called r-successor of a.

An ABox $\mathcal{A}$ is a finite set of concept and role assertions. The set $N_{I,\mathcal{A}}$ denotes the set of *individual names* occurring in ABox $\mathcal{A}$.

Definition 2.1.6 (Acyclic ABox)**.** Let $\mathcal{A}$ be an ABox. We call $\mathcal{A}$ is an acyclic ABox if there exists no individual a such that a occurs as r-successor of b and as r'-predecessor of c, for all $r, r' \in N_R$ and $b, c \in N_{I,\mathcal{A}}$. Otherwise, $\mathcal{A}$ is cyclic.

A knowledge base $\mathcal{K}$ consists of a TBox and an ABox ($\mathcal{K} = (\mathcal{T}, \mathcal{A})$). The set $sub(X)$ denotes the set of sub-concepts occurring in X, where X is a concept, ABox, TBox or a knowledge base. The set $N_{R,\mathcal{K}}$ denotes the set of *role names* occurring in $\mathcal{K}$.

The semantics of $\mathcal{EL}$-concepts are defined in terms of interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ [Baa+07].

Definition 2.1.7 (Interpretation)**.** An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$ of individuals and a mapping $\cdot^{\mathcal{I}}$ that maps

- every concept $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.
- every role $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.
- every individual name $a \in N_I$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

Given an interpretation $\mathcal{I}$, both syntax and semantics of $\mathcal{EL}$-concepts, TBox and ABox are summarized in Table 2.1. An interpretation $\mathcal{I}$ is said to satisfy a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is called a model of a TBox $\mathcal{T}$ if it satisfies all *GCIs* in $\mathcal{T}$. Let $C(a)$ be a concept assertion. Then an interpretation $\mathcal{I}$ is said to satisfy $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$. Let $r(a, b)$ be a role assertion. Then an interpretation $\mathcal{I}$ is said to satisfy $r(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. An interpretation $\mathcal{I}$ is called a model of an ABox $\mathcal{A}$ if it satisfies all concept and role assertions. An interpretation $\mathcal{I}$ is said to be a model of a knowledge base $\mathcal{K}$ if it satisfies both $\mathcal{T}$ and $\mathcal{A}$. Since we are only considering $\mathcal{EL}$, we sometimes write 'concept' and 'TBox' instead of '$\mathcal{EL}$-concept' and '$\mathcal{EL}$-TBox', respectively. *Subsumption* and *instance checking* are considered essential reasoning tasks that are generalized to computing the most specific concept of an individual. These two reasoning tasks in $\mathcal{EL}$ can be decided in polynomial time [BBL05].

Definition 2.1.8 (Instance checking and subsumption)**.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, let C, D be a $\mathcal{EL}$-concepts and a an individual name that occurs in $\mathcal{A}$, then:

- a is an instance of C w.r.t. $\mathcal{T}$ ($\mathcal{A} \models_{\mathcal{T}} C(a)$) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{K}$.
- C is subsumed by D w.r.t. $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$.

Table 2.1 The syntax and semantics of $\mathcal{EL}$-concept, TBox and ABox w.r.t. an interpretation $\mathcal{I}$.

name of constructor	Syntax	Semantics
concept name $A \in N_C$	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
role name $r \in N_R$	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
top-concept	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
GCI	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
concept definition	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
individual name	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
concept assertion	$A(a)$	$a^{\mathcal{I}} \in A^{\mathcal{I}}$
role assertion	$r(a,b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

2.2 The Most Specific Concept

Roughly speaking, the most specific concept of a given individual a is a generalization of a into a complex concept such that the concept is the most specific one of which an individual is an instance. The most specific concept of an individual a is defined based on subsumption and instance checking.

Definition 2.2.1 (Most specific concept)**.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and $a \in N_{I,\mathcal{A}}$ be an *individual name*. A concept C is the *most specific concept* of a w.r.t. $\mathcal{K}$ ($C = msc_{\mathcal{K}}(a)$) if the following statements hold:

- $\mathcal{K} \models C(a)$.
- $\mathcal{K} \models D(a)$ implies $C \sqsubseteq_{\mathcal{T}} D$.

As explained in [Baa03a] the msc does not always exist in $\mathcal{EL}$, in case we have a cyclic $\mathcal{A}$. However, if there is no msc, then one can approximate the msc concept by adding a bounded role-depth k of the concepts C and D. Then concept C is called a *role-depth bounded msc* of a w.r.t. $\mathcal{K}$.(k-$msc_{\mathcal{K}}(a)$). Both *msc* and *k-msc* are unique up to equivalence in $\mathcal{EL}$.

2.3 The Canonical Model and The Simulation Relation

The computation of the msc is based on the characterizations of instance checking and subsumption. Thus, such characterizations w.r.t. general TBoxes was introduced in [LW10b] by employing both *canonical models* and *simulation relations*. A homomorphism between the syntax trees of concepts was used to characterize subsumptions in the knowledge base with an empty TBox [BKM98]. Also, simulation relation plays the same role as homomorphism, and if applied on the canonical models, then it can also be used to characterize subsumptions. As shown in [ZT13], the decision of the existence of msc of an individual a w.r.t. $\mathcal{K}$ requires establishing a simulation relation between the canonical model and the constructed concept up to some bound k. Therefore, we will introduce the canonical model and simulation that we will use to decide the existence of msc.

Definition 2.3.1 (Canonical model of a concept)**.** Let $\mathcal{T}$ be a TBox, $\mathcal{I}$ a model of $\mathcal{T}$, and C a concept. The *canonical model* $\mathcal{I}_{C,\mathcal{T}}$ of C and $\mathcal{T}$ is defined as follows:

- $\Delta^{\mathcal{I}_{C,\mathcal{T}}} := \{d_C\} \cup \{d_{C'} \mid \exists r.C' \in sub(C) \cup sub(\mathcal{T})\}$.

- $A^{\mathfrak{I}_{C,\mathcal{T}}} := \{d_D \in \Delta^{\mathfrak{I}_{C,\mathcal{T}}} \mid D \sqsubseteq_{\mathcal{T}} A\}$, for all $A \in N_C$.
- $r^{\mathfrak{I}_{C,\mathcal{T}}} := \{(d_D, d_{D'}) \in \Delta^{\mathfrak{I}_{C,\mathcal{T}}} \mid D \sqsubseteq_{\mathcal{T}} \exists r.D' \text{ for } \exists r.D' \in sub(\mathcal{T})\}$, for all $r \in N_R$.

This definition can be extended to be the canonical model of a knowledge base.

Definition 2.3.2 (Canonical model of a knowledge base)**.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base. The canonical model $\mathfrak{I}_\mathcal{K}$ of $\mathcal{K}$ is defined as follows:

- $\Delta^{\mathfrak{I}_\mathcal{K}} := \{d_a \mid d_a \in N_{I,\mathcal{A}}\} \cup \{d_C \mid \exists r.C \in sub(\mathcal{K})\}$.
- $A^{\mathfrak{I}_\mathcal{K}} := \{d_a \in \Delta^{\mathfrak{I}_\mathcal{K}} \mid \mathcal{K} \models A(a)\} \cup \{d_C \in \Delta^{\mathfrak{I}_\mathcal{K}} \mid C \sqsubseteq_{\mathcal{T}} A\}$, for all $A \in N_C$.
- $r^{\mathfrak{I}_\mathcal{K}} := \{(d_a, d_D) \mid C \sqsubseteq D' \in sub(\mathcal{T}), \emptyset \models D' \sqsubseteq \exists r.D, a^{\mathfrak{I}_\mathcal{K}} \in C^{\mathfrak{I}_\mathcal{K}}\}$, for all $r \in N_R$.
- $a^{\mathfrak{I}_\mathcal{K}} := d_a$, for all $a \in N_{I,\mathcal{A}}$.

Let $\mathfrak{I}$ be an *interpretation*, $d \in \Delta^\mathfrak{I}$. Then, $(\mathfrak{I}, d)$ is said to be a *pointed interpretation*, where $\mathfrak{I}$ is rooted on d. The *simulation relations* are defined between pointed interpretations.

Definition 2.3.3 (Simulation)**.** Let $(\mathfrak{I}_1, d)$ and $(\mathfrak{I}_2, d')$ be interpretations. The binary relation $S \subseteq \Delta^{\mathfrak{I}_1} \times \Delta^{\mathfrak{I}_2}$ is called a *simulation* from $\mathfrak{I}_1$ to $\mathfrak{I}_2$ if all the following statements hold:

(S1) For every pair $(e_1, e_2) \in S$ it holds: $e_1 \in A^{\mathfrak{I}_1}$ implies $e_2 \in A^{\mathfrak{I}_2}$, for all $A \in N_C$.

(S2) if $(e_1, e_2) \in S$ and $(e_1, f_1) \in r^{\mathfrak{I}_1}$, then there exists some $f_2 \in \Delta^{\mathfrak{I}_2}$ such that $(f_1, f_2) \in S$ and $(e_2, f_2) \in r^{\mathfrak{I}_2}$.

A pointed interpretation $(\mathfrak{I}_1, d)$ is said to be *simulated* by a pointed interpretation $(\mathfrak{I}_2, d')$ (written as $(\mathfrak{I}_1, d) \lesssim (\mathfrak{I}_2, d')$) if there exists a simulation $S \subseteq \Delta^{\mathfrak{I}_1} \times \Delta^{\mathfrak{I}_2}$ with $(d, d') \in S$. Note that the relation $\lesssim$ is transitive and reflexive. A pointed interpretation $(\mathfrak{I}_1, d)$ is said to be *simulation equivalent* to a pointed interpretation $(\mathfrak{I}_2, d')$ (written as $(\mathfrak{I}_1, d) \simeq (\mathfrak{I}_2, d')$) if both $(\mathfrak{I}_1, d) \lesssim (\mathfrak{I}_2, d')$ and $(\mathfrak{I}_2, d') \lesssim (\mathfrak{I}_1, d)$ hold.

As shown [LW10b; ZT13], the canonical model has essential properties.

Lemma 2.3.4. *Let C be a concept and $\mathcal{T}$ be a TBox.*

(1) $d_E \in E^{\mathfrak{I}_{C,\mathcal{T}}}$, for all $d_E \in \Delta^{\mathfrak{I}_{C,\mathcal{T}}}$.

(2) $\mathfrak{I}_{C,\mathcal{T}}$ is a model of $\mathcal{T}$.

(3) $(\mathfrak{I}_{C,\mathcal{T}}, d_D) \simeq (\mathfrak{I}_{C',\mathcal{T}}, d_D)$, for all concepts C' and all $d_D \in \Delta^{\mathfrak{I}_{C,\mathcal{T}}} \cap \Delta^{\mathfrak{I}_{C',\mathcal{T}}}$.

(4) For all models $\mathfrak{I}$ of $\mathcal{T}$ and all $d \in \Delta^\mathfrak{I}$, the following statements are equivalent:

- *$d \in C^\mathfrak{I}$.*
- *$(\mathfrak{I}_{C,\mathfrak{I}}, d_C) \lesssim (\mathfrak{I}, d)$.*

(5) For every concept C and D, the following statements are equivalent:

- *$C \sqsubseteq_\mathcal{T} D$.*
- *$d_C \in D^{\mathfrak{I}_{C,\mathcal{T}}}$.*
- *$(\mathfrak{I}_{D,\mathcal{T}}, d_D) \lesssim (\mathfrak{I}_{C,\mathcal{T}}, d_C)$.*

Case (5) of the previous lemma allows us to give a characterization of the subsumption task, as shown in [LW10a]. If $C \sqsubseteq_\mathcal{T} D$, then $C^\mathfrak{I} \subseteq D^\mathfrak{I}$, for all models $\mathfrak{I}$ of $\mathcal{T}$. Hence, $C^\mathfrak{I} \in D^{\mathfrak{I}_{C,\mathcal{T}}}$, and $d_C \in D^{\mathfrak{I}_{C,\mathcal{T}}}$. Moreover, we have that $(\mathfrak{I}_{D,\mathcal{T}}, d_D) \lesssim (\mathfrak{I}_{C,\mathcal{T}}, d_C)$.

Similarly, in order to characterize the instance problem, consider the following lemma.

Lemma 2.3.5. *Let $\mathcal{K}$ be a knowledge base. $\mathfrak{I}_\mathcal{K}$ has the following properties:*

1. *$\mathfrak{I}_\mathcal{K}$ is a model of $\mathcal{K}$.*
2. *For every concept C, the following statements are equivalent:*

- $\mathcal{K} \models C(a)$.
- $d_a \in C^{\mathcal{I}_\mathcal{K}}$.

By definition of 2.1.8, we have that a is an instance of C w.r.t. $\mathcal{T}$ ($\mathcal{A} \models_\mathcal{T} C(a)$) iff $a^\mathcal{I} \in C^\mathcal{I}$ holds for all models $\mathcal{I}$ of $\mathcal{K}$. Therefore, if $\mathcal{I}_\mathcal{K}$ is the canonical model of $\mathcal{K}$, then for every concept C we have that $\mathcal{K} \models C(a)$ iff $d_a \in C^{\mathcal{I}_\mathcal{K}}$.

3 The Previous Approach for Computing The Msc

In this chapter, we revisit the previous approach [ZT13] for computing the msc. We introduce the tree unravelling of an interpretation $\mathcal{I}$. Then we present the construction of a concept from tree unravelling up to some bound k and show that it is sufficient to generate candidate concepts up to upper bound k. Then we show that an msc exists iff there exists a *simulation equivalent* relation between the pointed interpretation $(\mathcal{I}_\mathcal{K}, d_a)$ w.r.t. an individual a and the canonical model of one of the generated concepts.

3.1 The Tree Unravelling

Every interpretation $\mathcal{I}$ can be represented as a tree by taking an individual d as a root node and using as nodes all paths $\sigma = dr_1d_1r_2d_2r_3...$, where the nodes $\{d, d_1, d_2, ...\}$ are elements from $\Delta^\mathcal{I}$ and $\{r_1, r_2, ...\}$ are role names. (σ is a path in interpretation $\mathcal{I}$ if the individuals d_i and d_{i+1} are connected via $r^\mathcal{I}_{i+1}$, for all $i \in \mathbb{N}$). The nodes of the tree correspond to individual d_i in the path σ, and the edges of the tree correspond to role r_i in the path σ.

Definition 3.1.1 (Tree unravelling of an interpretation)**.** Let $\mathcal{I}$ be an interpretation and $d \in \Delta^\mathcal{I}$ w.r.t. N_C and N_R. The *tree unravelling* $\mathcal{I}_d$ of $\mathcal{I}$ in d is defined as follows:

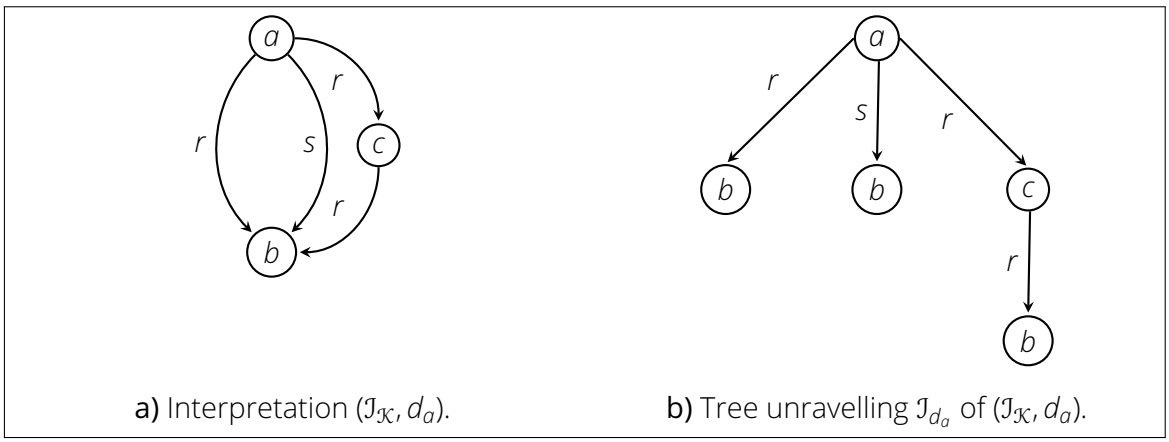
- $\Delta^{\mathcal{I}_d} := \{dr_1d_1r_2...r_nd_n \mid (d_i, d_{i+1}) \in r^\mathcal{I}_{i+1} \wedge 0 \leq i < n \wedge d_0 = d\}$.
- $A^{\mathcal{I}_d} := \{\sigma d' \mid \sigma d' \in \Delta^{\mathcal{I}_d} \wedge d' \in A^\mathcal{I}\}$, for all $A \in N_C$.
- $r^{\mathcal{I}_d} := \{(\sigma, \sigma r d') \mid (\sigma, \sigma r d') \in \Delta^{\mathcal{I}_d} \times \Delta^{\mathcal{I}_d}\}$, for all $r \in N_R$.

$|\sigma|$ denotes the number of role names occurring in σ, where $\sigma \in \Delta^\mathcal{I}$. Given $\sigma = dr_1d_1r_2...r_nd_n$, $tail(\sigma) = d_n$ is the last individual occuring in σ (also called the *tail* of σ).

Example 3. Consider The following knowledge base $\mathcal{K}$:
$\mathcal{K} = (\emptyset, \mathcal{A})$, with $\mathcal{A} = \{r(a, b), s(a, b), r(a, c), r(c, b)\}$.

Now we consider the pointed interpretation $(\mathcal{I}_\mathcal{K}, d_a)$ as shown in Figure 3.1a. We can see in Figure 3.1b that $(\mathcal{I}_\mathcal{K}, d_a)$ is unravelled into a finite tree. However, if the interpretation contains a loop, then the tree unravelling of the interpretation is infinite.

An interpretation $\mathcal{I}$ is unravelled into an infinite tree in case it contains a loop (cyclic path). Since the concept can only be constructed from a finite tree, therefore it is essential to unravel the tree up to depth k. $\mathcal{I}^k_{d_a}$ denotes the finite sub-tree with root d_a of the tree unravelling $\mathcal{I}_{d_a}$ containing all elements up to depth k.



a) Interpretation $(\mathcal{I}_{\mathcal{K}}, d_a)$. **b)** Tree unravelling $\mathcal{I}_{d_a}$ of $(\mathcal{I}_{\mathcal{K}}, d_a)$.

Figure 3.1 The pointed interpretation $(\mathcal{I}_{\mathcal{K}}, d_a)$ with no loop and the tree unravelling $\mathcal{I}_{d_a}$ of $(\mathcal{I}_{\mathcal{K}}, d_a)$ $\mathcal{I}_{d_a}$ constructed w.r.t. Example 3.

Definition 3.1.2 (Tree unravelling of an interpretation with a bounded role-depth)**.** Let $\mathcal{I}$ be an interpretation, $k \in \mathbb{N}$ and $d \in \Delta^{\mathcal{I}}$ w.r.t. N_C and N_R. The *tree unravelling* $\mathcal{I}_d$ of $\mathcal{I}$ in d up to bound k is defined as follows:

- $\Delta^{\mathcal{I}_d} := \{dr_1d_1r_2...r_kd_k \mid (d_i, d_{i+1}) \in r_{i+1}^{\mathcal{I}} \wedge 0 \leq i < k \wedge d_0 = d\}$.
- $A^{\mathcal{I}_d} := \{\sigma d' \mid \sigma d' \in \Delta^{\mathcal{I}_d} \wedge d' \in A^{\mathcal{I}} \wedge |\sigma| \leq k\}$, for all $A \in N_C$.
- $r^{\mathcal{I}_d} := \{(\sigma, \sigma r d') \mid (\sigma, \sigma r d') \in \Delta^{\mathcal{I}_d} \times \Delta^{\mathcal{I}_d} \wedge |\sigma| \leq k\}$, for all $r \in N_R$.

3.2 The Characteristic Concept w.r.t. A Tree Unravelling

Each finite tree interpretation can be translated into a so-called *characteristic concept*.

Definition 3.2.1 (Characteristic concept)**.** Let $(\mathcal{I}, d)$ be an interpretation. The *k-characteristic concept* $X^k(\mathcal{I}, d)$ is defined as follows[1]:

- $X^0(\mathcal{I}, d) = \prod\{A \in N_C \mid d \in A^{\mathcal{I}}\}$.
- $X^k(\mathcal{I}, d) = X^0(\mathcal{I}, d) \sqcap \prod\limits_{r \in N_{R,\mathcal{K}}} \prod\{\exists r.X^{k-1}(\mathcal{I}, d') \mid (d, d') \in r^{\mathcal{I}}\}$.

Example 4. Consider The following knowledge base $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{A}_1)$:

- $\mathcal{T}_1 = \{C \sqsubseteq \exists r.C\}$.
- $\mathcal{A}_1 = \{s(a, b), r(a, a), s(b, a), C(a), D(b)\}$.

As shown in Figure 3.2a the interpretation $(\mathcal{I}_{\mathcal{K}_1}, d_a)$ has two loops. Therefore, it can be unravelled into an infinite tree in Figure 3.2b. In order to translate it into a complex concept, it is essential to unravel the tree up to some bound, see Figure 3.2c. Thus, by Definition 3.2.1 we obtain the concept $X^2(\mathcal{I}_{\mathcal{K}_1}, d_a) = C \sqcap \exists r.(C \sqcap \exists r.C \sqcap \exists s.D) \sqcap \exists s.(D \sqcap \exists s.C)$. In the next section, we will show the previous approach [ZT13] to decide the existence of msc by employing *simulation relation* and *characteristic concept*.

3.3 Characterizing The Existence of The Msc

We recall some of the essential lemmas that are obtained from [ZT13]. First, we start with an important lemma, which establishes the relation between the tree unravelling $\mathcal{I}_{d_a}$ and $msc_{\mathcal{K}}(a)$.

[1] For a set N of concepts, $\prod N$ is abbreviation for $\sqcap_{C \in N} C$. If N is empty, then $\prod N$ is $\top$.

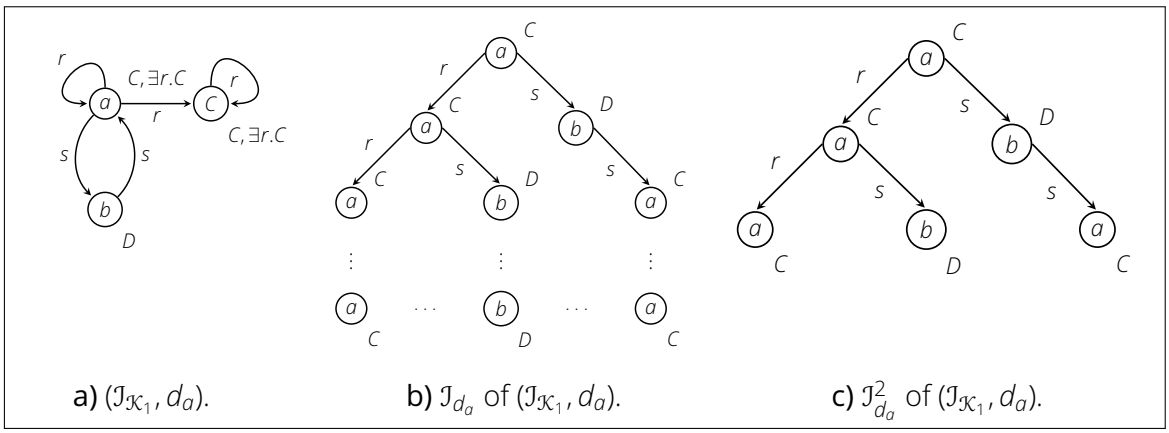


Figure 3.2 The interpretation $(\mathfrak{I}_{\mathcal{K}_1}, d_a)$ with two loops, the tree unravelling $\mathfrak{I}_{d_a}$ of $(\mathfrak{I}_{\mathcal{K}_1}, d_a)$ and the tree unravelling $\mathfrak{I}^2_{d_a}$ constructed w.r.t. Example 4.

Lemma 3.3.1. *([ZT13]) Let $\mathfrak{I}_{d_a}$ be the tree unravelling of $(\mathfrak{I}_{\mathcal{K}}, d_a)$ in d_a and C be k-msc$_{\mathcal{K}}(a)$. It holds that $\mathfrak{I}^k_{d_a} \lesssim (\mathfrak{I}_{C,\mathcal{T}}, d_C)$.*

To illustrate the previous lemma, if the concept C is a k-msc$_{\mathcal{K}}(a)$, then the canonical model of C simulates the tree unravelling up to k.

The decision of whether the concept C is an msc or not can be obtained from the following lemma.

Lemma 3.3.2. *([ZT13]) The concept C is the most specific concept of a w.r.t. $\mathcal{K}$ iff $(\mathfrak{I}_{\mathcal{K}}, d_a) \simeq (\mathfrak{I}_{C,\mathcal{T}}, d_C)$.*

Since the concept C can be constructed from a finite tree interpretation, it is required to define an upper bound to which the interpretation is unravelled. The upper bound is defined as follows.

Lemma 3.3.3. *([ZT13]) Let $m := max(\{rd(F) \mid F \in sub(\mathcal{K})\})$ and $n := |N_{I,\mathcal{A}}|$. If msc$_{\mathcal{K}}(a)$ exists, then $(\mathfrak{I}_{\mathcal{K}}, d_a) \lesssim \mathfrak{I}^{n^2+m+1}_{d_a}$.*

We are not only interested in computing the k-msc$_{\mathcal{K}}(a)$ with some bound k, but also in deciding the existence of msc$_{\mathcal{K}}(a)$ for all $k \in \mathbb{N}$ and compute msc$_{\mathcal{K}}(a)$ if it exists. This can be characterized by the following lemma.

Lemma 3.3.4. *([ZT13]) Let $\mathcal{K}$ be a knowledge base. msc$_{\mathcal{K}}(a)$ exists iff there exists a k such that the canonical model of $X^k(\mathfrak{I}_{\mathcal{K}}, d_a)$ w.r.t. $\mathcal{T}$ simulates $(\mathfrak{I}_{\mathcal{K}}, d_a)$.*

To decide the existence of the msc of an individual a w.r.t. $\mathcal{K}$,first we compute the upper bound k w.r.t. Lemma 3.3.3. Then the set M of L-characteristic concepts of $(\mathfrak{I}_{\mathcal{K}}, d_a)$ such that $M = \{X^L(\mathfrak{I}_{\mathcal{K}}, d_a) \mid L \leq k\}$. Finally, for each concept $C \in M$, C is the msc iff the canonical model of concept C w.r.t. $\mathcal{T}$ simulates $(\mathfrak{I}_{\mathcal{K}}, d_a)$ w.r.t. the individual a $((\mathfrak{I}_{\mathcal{K}}, d_a) \lesssim (\mathfrak{I}_{C,\mathcal{T}}, d_C))$. We illustrate this with the following example.

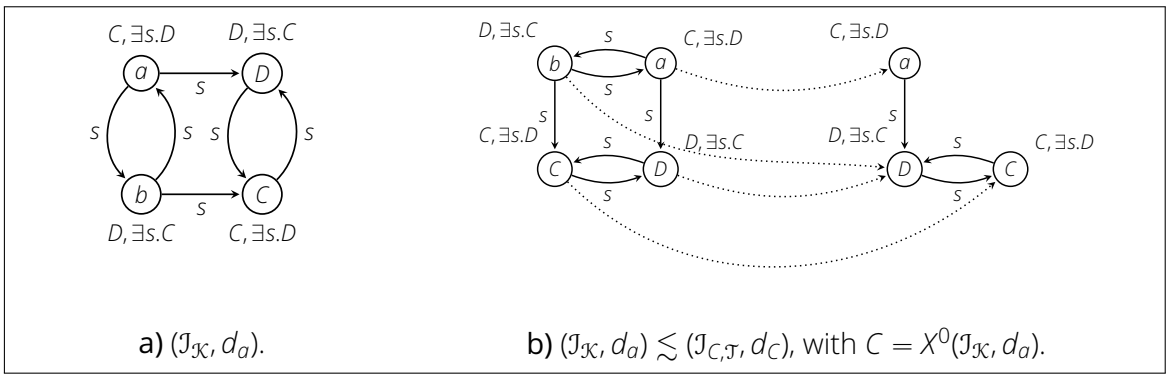


Figure 3.3 The interpretation $(\mathfrak{I}_{\mathcal{K}_1}, d_a)$, the simulation relation from $(\mathfrak{I}_{\mathcal{K}}, d_a)$ to $(\mathfrak{I}_{C,\mathcal{T}}, d_C)$, where $C = X^0(\mathfrak{I}_{\mathcal{K}}, d_a)$ w.r.t. Example 5.

Example 5. Consider the following knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$:

- $\mathcal{T} = \{C \sqsubseteq \exists s.D, D \sqsubseteq \exists s.C\}$.
- $\mathcal{A} = \{s(a, b), s(b, a), C(a), D(b)\}$.

In Figure 3.3 it is sufficient to generate concept with $k = 0$, as the canonical model of the concept $X^0(\mathfrak{I}_{\mathcal{K}}, d_a)$ w.r.t. $\mathcal{T}$ simulates $(\mathfrak{I}_{\mathcal{K}}, d_a)$. Thus $X^0(\mathfrak{I}_{\mathcal{K}}, d_a)$ is $msc_{\mathcal{K}}(a)$. Since computing the concept C could be done in exponential time in case the concept C is exponentially large, therefore in practice deciding the existence of the msc with the previous method could be done in exponential time.

In the next chapter, we introduce a new method for unravelling an interpretation. Then we employ this method to provide a more efficient way to decide the existence of the msc in a polynomial-time algorithm, which yields an efficient computation of the msc if it exists.

4 Towards Polynomial Computation

In this chapter, we revisit the sufficient condition from [Baa03a] for characterizing the existence of an msc in the terminological TBoxes. Then we introduce a new method for a tree unravelling of an interpretation $\mathcal{I}$. Moreover, we show the difference between the previous and the new method for the unravelling of an interpretation $\mathcal{I}$. Moreover, we show that the newly introduced method for tree unravelling has a tighter upper bound. Thus, the characteristic concept can be constructed w.r.t. this new method. Also, we use this newly introduced method in a polynomial approach for deciding the existence of the msc. Furthermore, we show the correctness and completeness of this Algorithm. Finally, we show that it decides the existence of the msc w.r.t. an individual in polynomial time.

The most specific concept always exists if the ABox is acyclic. However, it need not to exist in the case of cyclic ABox. As in Example 1 the most specific concept w.r.t. a does not exist in $\mathcal{K}_1$, due to the fact that $\mathcal{A}_1$ is cyclic. However, it exists in $\mathcal{K}_2$ even that $\mathcal{A}_2$ is cyclic. One can observe that not all cyclic paths cause this problem. The result from [Baa03a], which was defined w.r.t. terminological cycles, shows that the msc exists iff every cyclic path in $\mathcal{A}$ is simulated by at least one cyclic path in $\mathcal{T}$. However, we cannot ensure that it is still holds in the case of $\mathcal{EL}$-TBoxes. Intuitively, the goal is to extend this result to the case where we have a general $\mathcal{EL}$-TBoxes.

In Lemma 3.3.2, a concept C is the msc of an individual a iff there exists a *simulation equivalence* between $(\mathcal{I}_\mathcal{K}, d_a)$ and the canonical model of C w.r.t. $\mathcal{T}$, where the concept C is constructed from the tree unravelling of $(\mathcal{I}_\mathcal{K}, d_a)$ up to some bound k.

By Definition 2.3.2, if the interpretation $\mathcal{I}_\mathcal{K}$ contains cyclic paths, then these cyclic paths might generated by $\mathcal{T}$ or $\mathcal{A}$. Hence, $(\mathcal{I}_\mathcal{K}, d_a)$ w.r.t. an individual a contains all cyclic paths from which a is reachable. Therefore, it is sufficient to check, whether $(\mathcal{I}_\mathcal{K}, d_a)$ contains cycles or not. These cycles in the interpretation lead to the tree unravelling growing infinitely, while the tree unravelling of the Definition 3.1.1 cannot distinguish the existence of cycles in the interpretation. One direction to overcome such a problem is to use an upper bound k to which the tree is unravelled. Another direction is to define a new method for unravelling, a so-called least tree unravelling.

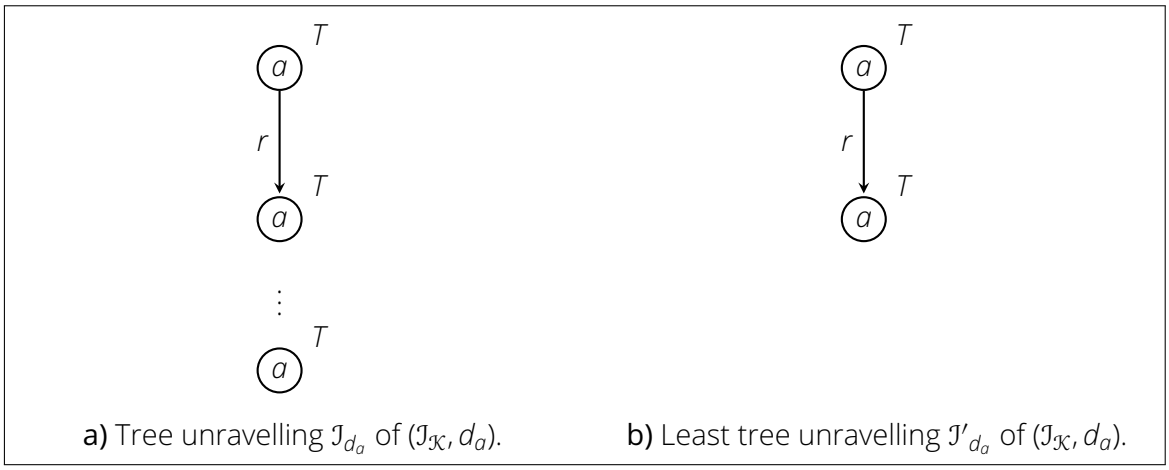
4.1 The Least Tree Unravelling

Roughly speaking, a least tree unravelling of an interpretation $\mathcal{I}$ is a tree unravelling, where each path $\sigma = dr_1d_1r_2...r_md_m$ contains only distinct individuals from $\Delta^\mathcal{I}$. If the $\Delta^\mathcal{I}$ is finite, then the tree constructed w.r.t. the least tree unravelling is also finite.

Definition 4.1.1 (Least tree unravelling of interpretation)**.** Let $\mathcal{I}$ be an interpretation and $d \in \Delta^\mathcal{I}$ w.r.t. N_C and N_R. The *least tree unravelling* $\mathcal{I}_d$ of $\mathcal{I}$ in d is defined as follows:

- $\Delta^{\mathfrak{I}'_d} := \{dr_1d_1r_2...r_nd_n \mid (d_i, d_{i+1}) \in r^{\mathfrak{I}}_{i+1} \wedge 0 \leq i < |\Delta^{\mathfrak{I}}| \wedge d_0 = d \wedge, \{d, d_1, ..., d_{n-1}\} \in \Delta^{\mathfrak{I}}$ *are distinct individuals*$\}$.
- $A^{\mathfrak{I}'_d} := \{\sigma d' \mid \sigma d' \in \Delta^{\mathfrak{I}'_d} \wedge d' \in A^{\mathfrak{I}}\}$, for all $A \in N_C$.
- $r^{\mathfrak{I}'_d} := \{(\sigma, \sigma r d') \mid (\sigma, \sigma r d') \in \Delta^{\mathfrak{I}'_d} \times \Delta^{\mathfrak{I}'_d}\}$, for all $r \in N_R$.

To illustrate the difference between the tree unravelling and the least tree unravelling, consider $(\mathfrak{I}_{\mathcal{K}_1}, d_a)$ defined w.r.t. $\mathcal{K}_1$ of Example 1. Since $\mathcal{A}_1$ is cyclic, the previous method for the tree unravelling grows infinitely. However, the tree constructed w.r.t. the least tree unravelling will have only tree depth equal to 1 (see Figure 4.1).



a) Tree unravelling $\mathfrak{I}_{d_a}$ of $(\mathfrak{I}_{\mathcal{K}}, d_a)$. **b)** Least tree unravelling $\mathfrak{I}'_{d_a}$ of $(\mathfrak{I}_{\mathcal{K}}, d_a)$.

Figure 4.1 The tree unravelling $\mathfrak{I}_{d_a}$ and the least tree unravelling $\mathfrak{I}'_{d_a}$ constructed w.r.t. $(\mathfrak{I}_{\mathcal{K}_1}, d_a)$ in Example 1.

By this definition, it is ensured that the tree constructed w.r.t. the least tree unravelling $\mathfrak{I}'_d$ of $\mathfrak{I}$ has a bounded depth k such that $k \leq |\Delta^{\mathfrak{I}}| + 1$. Hence, the tree is always finite, as each individual $d \in \Delta^{\mathfrak{I}'_d}$ appears only once in path $dr_1d_1...r_{n-1}d_{n-1}$. Given a path $\sigma = dr_1d_1...r_nd_n$, $tail(\sigma)$ could be an element that occurs only at the end of σ, or somewhere else in σ. One can observe that if an interpretation $\mathfrak{I}$ contains a cyclic path $\sigma = dr_1d_1...r_nd_n$, then $tail(\sigma) \in \{d, d_1, ...d_{n-1}\}$, otherwise the path is acyclic.

This characterization leads to checking the cycles in the interpretation $\mathfrak{I}$. If $\Delta^{\mathfrak{I}}$ is finite, then the concept $X^k(\mathfrak{I}, d)$ constructed w.r.t. the least tree unravelling always terminates. The concept $X(\mathfrak{I}, d)$ denotes the k-characteristic concept constructed w.r.t. the least tree unravelling, where $k = |\Delta^{\mathfrak{I}}| + 1$. If the interpretation $\mathfrak{I}$ is acyclic, the tree unravelling of $\mathfrak{I}$ and the least tree unravelling of $\mathfrak{I}$ have the same tree depth. Hence, the characteristic concept C constructed w.r.t. the least tree unravelling coincides with the characteristic concept C' constructed w.r.t. the tree unravelling. Thus, $(\mathfrak{I}_{C,\mathcal{T}}, d_C) \simeq (\mathfrak{I}_{C',\mathcal{T}}, d_{C'})$. Hence, C' is a $msc_{\mathcal{K}}(a)$ iff C is a $msc_{\mathcal{K}}(a)$.

Corollary 4.1.2. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, where $\mathcal{T}$ and $\mathcal{A}$ are acyclic. Let C and C' be the k-characteristic concepts constructed w.r.t. the tree unravelling of $(\mathfrak{I}_{\mathcal{K}}, d_a)$, with $k \in \mathbb{N}$ and the least tree unravelling of $(\mathfrak{I}_{\mathcal{K}}, d_a)$ respectively. Then C is a $msc_{\mathcal{K}}(a)$ iff C' is a $msc_{\mathcal{K}}(a)$.*

Proof. $\Rightarrow$ Suppose that C is a $msc_{\mathcal{K}}(a)$, then by the Lemma 3.3.2, $(\mathfrak{I}_{C,\mathcal{T}}, d_C) \simeq (\mathfrak{I}_{\mathcal{K}}, d_a)$. Since $\mathcal{T}$ and $\mathcal{A}$ are acyclic, then by the Definition 4.1.1 $\Delta^{\mathfrak{I}'_{d_a}} \subseteq \Delta^{\mathfrak{I}_{d_a}}$. For every path $\sigma = dr_1d_1...r_nd_n \in (\mathfrak{I}'_{d_a}, d_a)$ we have that $tail(\sigma) \notin \{d, d_1, ..., d_{n-1}\}$. Also, the tree unravelling is finite due to the fact that $\mathcal{T}$ and $\mathcal{A}$ are acyclic. Hence, $\Delta^{\mathfrak{I}'_{d_a}} = \Delta^{\mathfrak{I}_{d_a}}$. Therefore, the concept C' constructed w.r.t. the least tree unravelling and the concept C are identical. Moreover, $(\mathfrak{I}_{C,\mathcal{T}}, d_C) \simeq (\mathfrak{I}_{C',\mathcal{T}}, d_{C'})$. By Lemma 3.3.2, we obtain that $(\mathfrak{I}_{C',\mathcal{T}}, d_{C'}) \simeq (\mathfrak{I}_{\mathcal{K}}, d_a)$ and the concept C' is a $msc_{\mathcal{K}}(a)$.

$\Leftarrow$ The other direction can be proved analogously.

□

We still need to ensure whether Corollary 4.1.2 holds if $\mathcal{A}$ is cyclic. In the next section, we will introduce terminology that is essential in showing that the concept constructed w.r.t. the least tree unravelling coincides with the concept constructed w.r.t the tree unravelling. In the next section, we introduce a mechanism to convert an interpretation into a so-called $\mathcal{EL}$-graph.

4.2 An $\mathcal{EL}$-Graph

We start by defining interpretations in terms of graphs. Furthermore, we define a mechanism to convert a graph into an interpretation. Thus, the obtained interpretation can be used to construct a characteristic concept w.r.t. the least tree unravelling.

Definition 4.2.1 ($\mathcal{EL}$-Graph)**.** (From [Baa03a]) An $\mathcal{EL}$-graph is a directed graph $G = (V, E, Ind, Y)$, such that:

- V is a set of nodes.
- $E \subseteq V \times N_R \times V$ is a set of edges labeled by role names.
- $Y : V \to 2^{N_C}$ is a function that labels each node $v \in V$ with sets of concepts $L \subseteq N_C$.
- $Ind : V \to 2^{N_I}$ is a function that assigns each node $v \in V$ to a set of individual names.

We denote the root node of G by v_0. A set $pe(E, v)$ is the set of all edges with a node v as a predecessor.

Definition 4.2.2 (Sub-graph)**.** Let $G_1 = (V_1, E_1, Ind_1, Y_1)$ be an $\mathcal{EL}$-graph. The $\mathcal{EL}$-graph $G_2 = (V_2, E_2, Ind_2, Y_2)$ is said to be a sub-graph of G_1 ($G_2 \subseteq G_1$) iff the following statements hold:

- $V_2 \subseteq V_1$.
- $E_2 \subseteq E_1$ such that for every $e = (v, r, d) \in E_2$ we have $v, d \in V_2$.
- $Y_2(v) \subseteq Y_1(v)$, for all $v \in V_2$.
- $Ind_2(v) \subseteq Ind_1(v)$, for all $v \in V_2$.

The interpretation $\mathfrak{I}$ can be translated into an $\mathcal{EL}$-graph by the following definition.

Definition 4.2.3 (Interpretation as a graph)**.** (From [Baa03a]) Let $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$ be an interpretation and d be an element in $\Delta^{\mathfrak{I}}$. The graph[1] $G_{\mathfrak{I}} = (V_{\mathfrak{I}}, E_{\mathfrak{I}}, Ind_{\mathfrak{I}}, Y_{\mathfrak{I}})$ of the interpretation $\mathfrak{I}$ is defined as follows:

- $V_{\mathfrak{I}}$ are the elements of $\Delta^{\mathfrak{I}}$, with $v_0 = d$.
- $E_{\mathfrak{I}} = \{(x, r, y) \mid r \in N_R, (x, y) \in r^{\mathfrak{I}}\}$.
- $Y_{\mathfrak{I}}(v) = \{L \in N_C \mid v \in L^{\mathfrak{I}}\}$, for all $v \in \Delta^{\mathfrak{I}}$.
- $Ind_{\mathfrak{I}}(v) = \{a \in N_I \mid v \in a^{\mathfrak{I}}\}$, for all $v \in \Delta^{\mathfrak{I}}$.

Given a graph $G = (V, E, Ind, Y)$, a node $v' \in V$ is said to be directly reachable from a node $v \in V$ (denoted by $reach_{direct}(v, v')$), if there exists a path σ from v to v' and $|\sigma| = 1$ ($|\sigma|$ denotes the number of role names occurring in σ). More precisely, $v' \in V$ is directly reachable from a node $v \in V$ iff there exists an edge $(v, r, v') \in E$. A node $v' \in V$ is said to be reachable from a node $v \in V$ (denoted by $reach(v, v')$), if there exists a path $\sigma = vr_1 \ldots r_n v'$ from v to v'.

A path $\sigma = v_0 r_1 v_1 \ldots r_n v_n$ is said to be reachable from a node $v \in V$, if the node v_0 is reachable from v (denoted by $reach(v, \sigma)$). We denote by $head(\sigma)$ the first element of the path $\sigma = v_0 r_1 v_1 \ldots r_n v_n$, while $tail(\sigma)$ denotes the last element of the path $\sigma = v_0 r_1 v_1 \ldots r_n v_n$.

Proposition 4.2.4. *([LPP12]) Let $G = (V, E, Ind, Y)$ be an $\mathcal{EL}$-graph, v, v' nodes of V. It is decidable in polynomial time whether v' is reachable from v.*

[1] We write graph to refer to $\mathcal{EL}$-graph.

In order to construct a characteristic concept, it is essential to provide a mechanism to translate a graph G into an interpretation $\mathfrak{I}_G$, which is defined as follows.

Definition 4.2.5. [Interpretation from graph] Let $G = (V, E, Ind, Y)$ be a graph, $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a knowledge base. The interpretation $\mathfrak{I}_G$ w.r.t. N_R, N_C and $N_{I,\mathcal{A}}$ is defined as follows:

- $\Delta^{\mathfrak{I}_G} = V$.
- $A^{\mathfrak{I}_G} = \{d_v \mid A \in Y(v) \wedge v \in V\}$, for all $A \in N_C$.
- $r^{\mathfrak{I}_G} = \{(d_v, d_{v'}) \mid (v, r, v') \in E\}$, for all $r \in N_R$.
- $a^{\mathfrak{I}_G} = \{d_a \mid a \in Ind(v) \wedge v \in V\}$, for all $a \in N_{I,\mathcal{A}}$.

As we are interested in characterizing the cycles of $\mathfrak{I}$, we need to distinguish between the cyclic paths and the acyclic paths of $G_\mathfrak{I}$. This can be obtained from the following definition.

Definition 4.2.6 (Cyclic path)**.** Let $G_\mathfrak{I} = (V_\mathfrak{I}, E_\mathfrak{I}, Ind_\mathfrak{I}, Y_\mathfrak{I})$ be a graph. A cyclic path G_σ is a path of $G_\mathfrak{I}$ such that:

- $G_\sigma = \{x_0 r_1 x_1 ... r_n x_n \mid (x_i, r_i, x_{i+1}) \in E_\mathfrak{I} \wedge x_n \in \{x_0, x_1, ..., x_{n-1}\} \wedge 0 \leq i < n \wedge n \leq |V_\mathfrak{I}|\}$.

$|G_\sigma|$ denotes the number of role names occurring in G_σ. The set G_{cycles} denotes the set of all cyclic paths G_σ of $G_\mathfrak{I}$.

To be able to distinguish between the cycles of $\mathfrak{I}$ that are generated from $\mathcal{T}$ and $\mathcal{A}$, we need to construct sub-graphs $G'_{\mathfrak{I},\mathcal{A}}$ and $G''_{\mathfrak{I},\mathcal{T}}$ of $G_\mathfrak{I}$.

Definition 4.2.7 (ABox graph)**.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, a an individual, $G_\mathfrak{I} = (V_\mathfrak{I}, E_\mathfrak{I}, Ind_\mathfrak{I}, Y_\mathfrak{I})$ be the graph of $(\mathfrak{I}_\mathcal{K}, d_a)$. The ABox graph $G_{\mathfrak{I},\mathcal{A}} = (V_{\mathfrak{I},\mathcal{A}}, E_{\mathfrak{I},\mathcal{A}}, Ind_{\mathfrak{I},\mathcal{A}}, Y_{\mathfrak{I},\mathcal{A}})$ is a sub-graph of $G_\mathfrak{I}$ such that:

- $V_{\mathfrak{I},\mathcal{A}} = \{v \in V_\mathfrak{I} \mid Ind(v) \in N_{I,\mathcal{A}}\}$.
- $E_{\mathfrak{I},\mathcal{A}} = \{(x, r, y) \mid (x, y) \in r^{(\mathfrak{I}_\mathcal{K}, d_a)} \wedge x, y \in V_{\mathfrak{I},\mathcal{A}}\}$.
- $Y_{\mathfrak{I},\mathcal{A}}(v) = \{L \in N_C \mid v \in L^{(\mathfrak{I}_\mathcal{K}, d_a)}\}$, for all $v \in V_{\mathfrak{I},\mathcal{A}}$.
- $Ind_{\mathfrak{I},\mathcal{A}}(v) = Ind_\mathfrak{I}(v)$, for all $v \in V_{\mathfrak{I},\mathcal{A}}$.

$V_\mathcal{A}$ denotes the set of nodes that are *generated by* $\mathcal{A}$.

The graph $G_{\mathfrak{I},\mathcal{T}}$ can be defined similarly.

Definition 4.2.8 (TBox graph)**.** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, a an individual, $G_\mathfrak{I} = (V_\mathfrak{I}, E_\mathfrak{I}, Ind_\mathfrak{I}, Y_\mathfrak{I})$ be the graph of $(\mathfrak{I}_\mathcal{K}, d_a)$. The TBox graph $G_{\mathfrak{I},\mathcal{T}} = (V_{\mathfrak{I},\mathcal{T}}, E_{\mathfrak{I},\mathcal{T}}, Ind_{\mathfrak{I},\mathcal{T}}, Y_{\mathfrak{I},\mathcal{T}})$ is a sub-graph of $G_\mathfrak{I}$ such that:

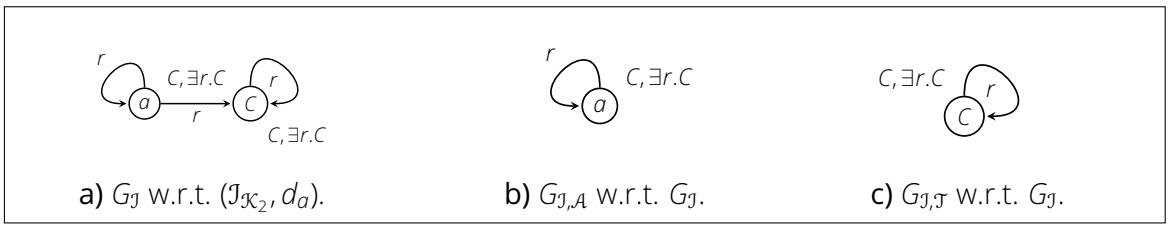
- $V_{\mathfrak{I},\mathcal{T}} = \{v \in V_\mathfrak{I} \mid Ind(v) \notin N_{I,\mathcal{A}}\}$.
- $E_{\mathfrak{I},\mathcal{T}} = \{(x, r, y) \mid (x, y) \in r^{(\mathfrak{I}_\mathcal{K}, d_a)} \wedge x, y \in V_{\mathfrak{I},\mathcal{T}}\}$.
- $Y_{\mathfrak{I},\mathcal{T}}(v) = \{L \in N_C \mid v \in L^{(\mathfrak{I}_\mathcal{K}, d_a)}\}$, for all $v \in V_{\mathfrak{I},\mathcal{T}}$.
- $Ind_{\mathfrak{I},\mathcal{T}}(v) = Ind_\mathfrak{I}(v)$, for all $v \in V_{\mathfrak{I},\mathcal{T}}$.

$V_\mathcal{T}$ denotes the set of nodes that are *generated by* $\mathcal{T}$.

We observe that $G_{\mathfrak{I},\mathcal{A}}$ and $G_{\mathfrak{I},\mathcal{T}}$ are unique w.r.t. the same model $(\mathfrak{I}_\mathcal{K}, d_a)$. Moreover, it is clear to see that $G_{\mathfrak{I},\mathcal{T}}$ and $G_{\mathfrak{I},\mathcal{A}}$ are sub-graphs of $G_\mathfrak{I}$, where $G_{\mathfrak{I},\mathcal{T}}$ and $G_{\mathfrak{I},\mathcal{A}}$ are disjoint (see Figure 4.2).

From the fact that $G_{\mathfrak{I},\mathcal{A}}$ and $G_{\mathfrak{I},\mathcal{T}}$ are sub-graphs of $G_\mathfrak{I}$, where $G_\mathfrak{I}$ is the graph of $(\mathfrak{I}_\mathcal{K}, d_a)$, one can observe that adding all paths $\sigma \in G_\mathfrak{I}$ to $G_{\mathfrak{I},\mathcal{A}}$ that are reachable from any node v in $G_{\mathfrak{I},\mathcal{A}}$ allows us to obtain back $G_\mathfrak{I}$.

Definition 4.2.9 (Canonical graph)**.** Let $G_1 = (V_1, E_1, Ind_1, Y_1)$ be the graph of $(\mathfrak{I}_\mathcal{K}, d_a)$, $G_{\mathfrak{I},\mathcal{A}}$ the ABox graph of G_1, $G_2 = (V_2, E_2, Ind_2, Y_2)$ a sub-graph of $G_{\mathfrak{I},\mathcal{A}}$. We define as *canonical graph* of G_2 w.r.t. G_1 as the graph $G_{G_2,G_1} = (V_{G_2,G_1}, E_{G_2,G_1}, Ind_{G_2,G_1}, Y_{G_2,G_1})$ such that:



a) $G_{\mathcal{I}}$ w.r.t. $(\mathfrak{I}_{\mathcal{K}_2}, d_a)$. **b)** $G_{\mathcal{I},\mathcal{A}}$ w.r.t. $G_{\mathcal{I}}$. **c)** $G_{\mathcal{I},\mathcal{T}}$ w.r.t. $G_{\mathcal{I}}$.

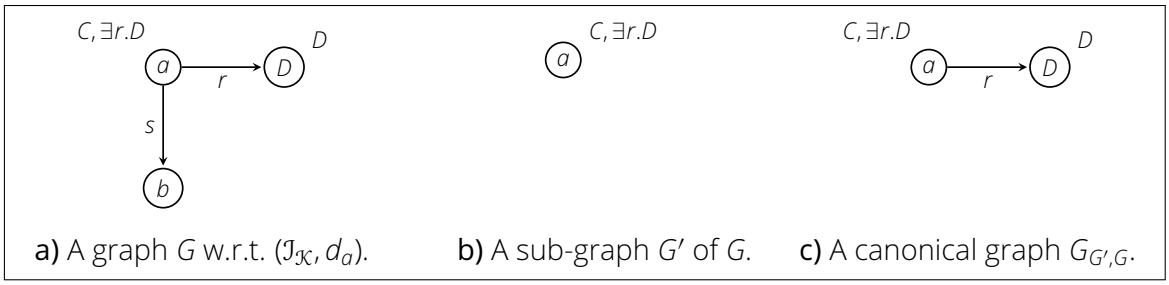
Figure 4.2 The graph $G_{\mathcal{I}}$ defined w.r.t. $(\mathfrak{I}_{\mathcal{K}_2}, d_a)$ of Example 1. The graphs $G_{\mathcal{I},\mathcal{A}}$ and $G_{\mathcal{I},\mathcal{T}}$ w.r.t. $G_{\mathcal{I}}$.

- $V_{G_2,G_1} = V_2 \cup \{v' \in V_1 \mid v' \text{ is reachable from } v \wedge Ind_1(v') \notin N_{I,\mathcal{A}}, v \in V_2\}$.
- $E_{G_2,G_1} = \{(x,r,y) \in E_1 \mid x,y \in V_{G_2,G_1}\}$.
- $Y_{G_2,G_1}(v) = Y_1(v)$, for all $v \in V_{G_2,G_1}$.
- $Ind_{G_2,G_1}(v) = Ind_1(v)$, for all $v \in V_{G_2,G_1}$.

Example 6. Consider The following knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$:

- $\mathcal{T} = \{C \sqsubseteq \exists r.D\}$.
- $\mathcal{A} = \{s(a,b), C(a)\}$.

The previous definition adds only those nodes and edges that are generated by $G_{\mathcal{I},\mathcal{T}}$ (see Figure 4.3 of Example 6). As a consequence of the previous definition, we obtain that the canonical graph $G_{G_{\mathcal{I},\mathcal{A}},G_{\mathcal{I}}}$ of $G_{\mathcal{I},\mathcal{A}}$ w.r.t. $G_{\mathcal{I}}$ gives us back $G_{\mathcal{I}}$. Thus, we have that $(G_{G_{\mathcal{I},\mathcal{A}},G_{\mathcal{I}}}, v_0)$ is identical to $(G_{\mathcal{I}}, v_0)$. Moreover, if $\mathcal{A}$ is acyclic, then constructing a concept C w.r.t. the least tree unravelling of $\mathfrak{I}_{G_{G_{\mathcal{I},\mathcal{A}},G_{\mathcal{I}}}}$ we have that by the Corollary 4.1.2 $(\mathfrak{I}_{C,\mathcal{T}}) \simeq (\mathfrak{I}_{\mathcal{K}}, d_a)$.



a) A graph G w.r.t. $(\mathfrak{I}_{\mathcal{K}}, d_a)$. **b)** A sub-graph G' of G. **c)** A canonical graph $G_{G',G}$.

Figure 4.3 A graph G constructed w.r.t. $(\mathfrak{I}_{\mathcal{K}}, d_a)$, a sub-graph G' of G, and the canonical graph $G_{G',G}$ w.r.t. G' and G.

From the definitions 4.2.2, 4.2.9 and 4.2.7, we can obtain the following Lemma that characterizes the relation between two sub-graphs of the ABox graph $G_{\mathcal{A}}$ constructed w.r.t. the same graph.

Lemma 4.2.10. *Let $G = (V, E, Ind, Y)$ be the $\mathcal{EL}$-graph of $(\mathfrak{I}_{\mathcal{K}}, d_a)$, $G_1 = (V_1, E_1, Ind_1, Y_1), G_2 = (V_2, E_2, Ind_2, Y_2)$ be two sub-graphs of the ABox graph $G_{\mathcal{A}}$ such that $G_1 \subseteq G_2$. Then $G_{G_1,G} = (V_{G_1,G}, E_{G_1,G}, Ind_{G_1,G}, Y_{G_1,G}) \subseteq G_{G_2,G} = (V_{G_2,G}, E_{G_2,G}, Ind_{G_2,G}, Y_{G_2,G})$.*

Proof. Assume that $G_1 \subseteq G_2$. Then by the Definition 4.2.2, we have the following:

- $V_1 \subseteq V_2$.
- $E_1 \subseteq E_2$.
- $Y_1(v) \subseteq Y_2(v)$, for all $v \in V_1$.
- $Ind_1(v) \subseteq Ind_2(v)$, for all $v \in V_1$.

Also, from the construction of the Definition 4.2.9, we have that the canonical graph $G_{G_1,G}$ is a sub-graph of G such that:

- $V_{G_1,G}$ contains all nodes from V_1 together with all nodes $v \in V_\mathfrak{T}$ such that v is reachable from a node $v' \in V_1$.
- $E_{G_1,G}$ contains all edges $e = (x, r, y) \in E$ such that $x, y \in V_{G_1,G}$.
- $Y_{G_1,G}(v) \subseteq Y(v)$ such that $v \in V_{G_1,G}$.
- $Ind_{G_1,G}(v) \subseteq Ind(v)$ such that $v \in V_{G_1,G}$.

The canonical graph $G_{G_2,G}$ is constructed analogously. Then by construction of the Definition 4.2.9 if there exists a node v such that $v \in V_{G_1,G} \wedge v \notin V_1$, then v must be reachable from a node $v' \in V_1$ where $v' \in V_1$. Since $V_1 \subseteq V_2$, then the node v' is also in V_2, and v must be in $G_{G_2,G}$ and $v \notin V_2$. Therefore, $V_{G_1,G}$ is a subset of $V_{G_2,G}$ ($V_{G_1,G} \subseteq V_{G_2,G}$).

Similarly, if there exists an edge $e = (x, r, y) \in E_{G_1,G}$, then $e = (x, r, y) \in E_{G_2,G}$. Due to the fact that $x, y \in V_{G_1,G}$ and $x, y \in V_{G_2,G}$. From $V_{G_1,G} \subseteq V_{G_2,G}$ we have that $Y_{G_1,G}$ is also a subset of $Y_{G_2,G}$ ($Y_{G_1,G} \subseteq Y_{G_2,G}$). □

We can also characterize the edges that are added in the canonical graphs of two sub-graphs of the ABox graph $G_\mathcal{A}$ w.r.t. the same graph $G_\mathfrak{J}$. This can be characterized as follows.

Lemma 4.2.11. *Let $G = (V, E, Ind, Y)$ be the $\mathcal{EL}$-graph of $(\mathfrak{I}_\mathfrak{K}, d_a)$, $V_\mathfrak{T}$ be the set of nodes in G that are generated by $\mathfrak{T}$, $G_1 = (V_1, E_1, Ind_1, Y_1)$, $G_2 = (V_2, E_2, Ind_2, Y_2)$ be two sub-graphs of the ABox graph $G_\mathcal{A}$. For $v_1 \in V_{G_1,G}, v_2 \in V_{G_2,G}$ if $Y_{G_1,G}(v_1) \subseteq Y_{G_2,G}(v_2)$, $(v_1, r, v) \in E_{G_1,G}$ and $v \in V_\mathfrak{T}$, then $(v_2, r, v) \in E_{G_2,G}$ where $G_{G_1,G} = (V_{G_1,G}, E_{G_1,G}, Ind_{G_1,G}, Y_{G_1,G})$ and $G_{G_2,G} = (V_{G_2,G}, E_{G_2,G}, Ind_{G_1,G}, Y_{G_2,G})$ are the canonical graphs of G_1 and G_2 w.r.t. G.*

Proof. We assume that ontology is in normal form. An ontology is in a *normal form* if every existential restriction occurs in axioms of the following form:

- $\exists r.A \sqsubseteq B$. Or,
- $A \sqsubseteq \exists r.B$.

By construction of the Definition 4.2.9 we have that all nodes $v \in V_\mathfrak{T}$ that are added to $G_{G_1,G}$ must be reachable from a node $v_1 \in V_1$. Therefore, there must exists an edge $(v_1, r, v) \in E_{G_1,G}$, where $v \in V_\mathfrak{T}$. Also by construction of the Definition 2.3.2, the edge $(v_1, r, v) \in E_{G_1,G}$ exists in $G_{G_1,G}$ iff $(v_1, v) \in r^{\mathfrak{J}_\mathfrak{K}}$. Moreover, we can obtain the set of concept names M such that every $C \in M$ we have $v_1 \in C^{\mathfrak{J}_\mathfrak{K}}$. Similarly, we can obtain the set of concepts M' such that every $C' \in M'$ we have $v_2 \in C'^{\mathfrak{J}_\mathfrak{K}}$. If $Y_{G_1,G}(v_1) \subseteq Y_{G_2,G}(v_2)$, then $M \subseteq M'$. Hence, we have that every concept $C \in M$, we have that $v_2 \in C^{\mathfrak{J}_\mathfrak{K}}$. Hence, as a consequence of the construction of the Definition 2.3.2,we have that $(v_2, v) \in r^{\mathfrak{J}_\mathfrak{K}}$ and $(v_1, v) \in r^{\mathfrak{J}_\mathfrak{K}}$, for the same role name r. Hence, the node $v \in V_\mathfrak{T}$ must be reachable from the node $v_2 \in V_{G_2,G}$. Therefore, there must exists an edge $(v_2, r, v) \in E_{G_2,G}$. □

As graphs are used to represent the interpretations, then it is necessary to define the simulation explicitly in terms of graphs.

4.3 The Simulation Over Graphs

Definition 4.3.1. [Simulation over graphs] Let $G_1 = (V_1, E_1, Ind_1, Y_1)$, $G_2 = (V_2, E_2, Ind_2, Y_2)$ be $\mathcal{EL}$-graphs, $v \in V_1$ be a node, $v' \in V_2$ be a node. A binary relation $S \subseteq V_1 \times V_2$ is called a *simulation from* (G_1, v) *to* (G_2, v') if all the following conditions are satisfied:

(S1) If $(v_1, v_2) \in S$, then $Y_1(v_1) \subseteq Y_2(v_2)$.

(S2) If $(v_1, v_2) \in S$, and $(v_1, r, v_1') \in E_1$, then there exists some $v_2' \in V_2$ such that $(v_1', v_2') \in S$, and $(v_2, r, v_2') \in E_2$.

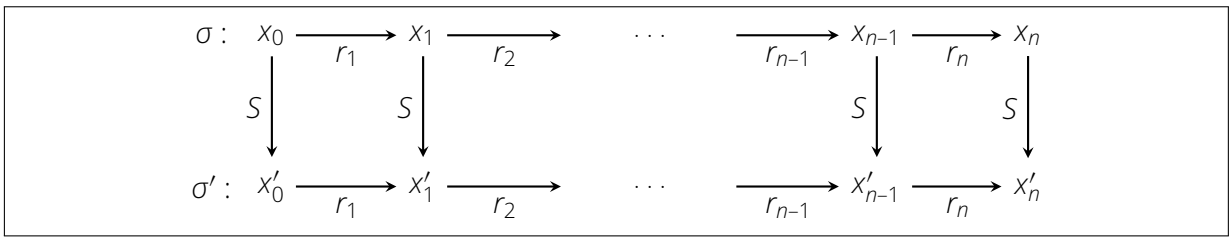


Figure 4.4 The simulation S from path σ to σ' .

We write $S : (G_1, v) \lesssim (G_2, v')$ to express that there exists a simulation S from G_1 to G_2 such that $(v, v') \in S$. We write $S : (G_1, v) \simeq (G_2, v')$ to express that there exists a simulation equivalence between G_1 and G_2 ($(G_1, v) \lesssim (G_2, v')$ and $(G_2, v') \lesssim (G_1, v)$).[2]

It is easy to see that the set of all simulations from G_1 to G_2 is closed under arbitrary unions, if G_1 and G_2 are finite. Consequently, there always exists a greatest simulation from G_1 to G_2. If G_1 and G_2 are finite, then this greatest simulation can be computed in polynomial time [HM01]. As a consequence of this fact, we obtain the following proposition.

Proposition 4.3.2. *([Baa03b]) Let G_1, G_2 be two finite $\mathcal{EL}$-graphs, v a node of G_1 and v' a node of G_2. It is decidable in polynomial time whether there exists a simulation $S : (G_1, v) \lesssim (G_2, v')$ such that $(v, v') \in S$.*

Definition 4.3.3 (Simulation over paths)**.** Let G be a graph constructed w.r.t. an interpretation $\mathfrak{I}$, σ, σ' be two paths occurring in G. A path $\sigma = x_0 r_1 \ldots$ is said to be simulated by $\sigma' = x'_0 r_1 x'_0 r_2 \ldots$ iff there exists a simulation S such that $(x_i, x'_i) \in S$, for all $0 \leq i$ and $x_0 = x'_0$. We write $S : \sigma \lesssim \sigma'$ to express that there exists a simulation S from σ to σ' (see Figure 4.4).

Definition 4.3.4 (Simulation over edges)**.** Let $G = (V, E, Ind, Y)$ be an $\mathcal{EL}$-graph, $e = (v, r, d), e' = (v', r, d')$ be two edges occurring in E. e is said to be simulated by e' ($e \lesssim e'$) if the following conditions are satisfied:

- $Y(v) \subseteq Y(v')$.
- There exists a simulation S such that $(d, d') \in S$.

4.4 Computing The Msc w.r.t. The Least Tree Unravelling

Now we are ready to construct the most specific concept w.r.t. the least tree unravelling. Instead of generating all the candidate concepts up to upper bound k and check whether the *simulation equivalence* holds between the canonical model of one of the candidates w.r.t. $\mathcal{T}$ and $(\mathfrak{I}_\mathcal{K}, d_a)$, we employ the Lemmas 3.3.2 and 4.1.2 to decide the existence of the msc w.r.t. an individual.

Intuitively, we construct a sub-interpretation $(\mathfrak{I}', d_a)$ of $(\mathfrak{I}_\mathcal{K}, d_a)$ up to equivalence, such that $(\mathfrak{I}', d_a) \lesssim (\mathfrak{I}_\mathcal{K}, d_a)$ and $(\mathfrak{I}_\mathcal{K}, d_a) \lesssim (\mathfrak{I}', d_a)$. We construct a sub-graph $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ of ABox graph $G_\mathcal{A}$ by recursively copying nodes and concepts related to them from the graph $canM = (V, E, Ind, Y)$ of $(\mathfrak{I}_\mathcal{K}, d_a)$ starting from the root node v_0. v_0 is the node that represents the individual a. We define the construction of $mscG$ in terms of an algorithm. The algorithms that we will be used in the following sections of this chapter, will be captured in Chapter 5.

[2] We may just write $(G_1, v) \lesssim (G_2, v')$ instead of $S : (G_1, v) \lesssim (G_2, v')$, if the simulation relation is not important in the context.

4.4.1 Deciding The Existence of The Msc

From Now we use the graph $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ to represent the graph that we want to construct, the graph $canM = (V, E, Ind, Y)$ to represent the graph of $(\mathfrak{I}_{\mathcal{K}}, d_a)$ and the graph $temp_t = (V_t, E_t, Ind_t, Y_t)$ to be the canonical graph of $mscG$ w.r.t. $canM$ after the termination of the recursive call of the Algorithm 2.

We start by taking $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and the individual a as inputs. Then $canM$ is constructed w.r.t. $(\mathfrak{I}_{\mathcal{K}}, d_a)$.

We initialize $mscG_0 = (V_{mscG_0}, E_{mscG_0}, Ind_{mscG_0}, Y_{mscG_0})$ as follows:

- $V_{mscG_0} = (v_0)$, where v_0 is the root node of $canM$.
- $E_{mscG_0} = \emptyset$.
- $Y_{mscG_0}(v_0) = Y(v_0)$.
- $Ind_{mscG_0}(v_0) = a$.

Then, recursively we visit every node $v \in V_{\mathcal{A}}$ (recall the Definition 4.2.7) and add the edges if needed. More precisely, for $0 \leq i \leq |V_{\mathcal{A}}|$, we do the following:

- we build $temp_i = (V_i, E_i, Ind_i, Y_i)$ to be the canonical graph of $mscG_i$ w.r.t. $canM$.
- for each edge $(v, r, d) \in pe(E, v)$, we check whether there exists an edge $(v, r, d') \in pe(E_i, v)$ such that (v, r, d) is simulated by (v, r, d').

We can define $mscG_i = (V_{mscG_i}, E_{mscG_i}, Ind_{mscG_i}, Y_{mscG_i})$, where $i > 0$ as follows:

- $V_{mscG_i} = V_{mscG_{i-1}} \cup \{Ind(d) \in N_{I,\mathcal{A}} \mid$ there exists some $v \in V_{mscG_{i-1}}$ and for all $(v, r, d') \in pe(E_{i-1}, v)$ such that $(v, r, d) \in pe(E, v)$ w.r.t. canM $\wedge (v, r, d) \not\lesssim (v, r, d')\}$.
- $E_{mscG_i} = \{(v, r, d) \in pe(E, v) \mid v, d \in V_{mscG_i}\}$.
- $Y_{mscG_i}(v) = Y(v)$, for all $v \in V_{mscG_i}$.
- $Ind_{mscG_i}(v) = Ind(v)$, for all $v \in V_{mscG_i}$.

In the previous definition, only nodes that are contained in $V_{\mathcal{A}}$ are added to V_{mscG_i}. Since the number of nodes of the graph G is finite, then also the size of $V_{\mathcal{A}}$ is finite. Also, the number of edges in E is finite. Therefore, we ensure that the Algorithm terminates.

After Algorithm terminates, we obtain that $mscG$ is a sub-graph of ABox graph $G_{\mathcal{A}}$. The reason is that $mscG$ contains only nodes from $V_{\mathcal{A}}$. Moreover, we observe that there exists always a simulation equivalence between $(temp_t, v_0)$ and $(canM, v_0)$. This can be characterized by the following lemmas.

Lemma 4.4.1. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, $canM = (V, E, Ind, Y)$ be the graph of $(\mathfrak{I}_{\mathcal{K}}, d_a)$, and $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ be a graph constructed after the termination of Algorithm 1. It holds that $(canM, v_0) \lesssim (temp_t, v_0)$, where $temp_t = (V_t, E_t, Ind_t, Y_t)$ is the canonical graph of mscG w.r.t. canM.*

Proof. Let $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}}, Ind_{\mathcal{A}}, Y_{\mathcal{A}})$ be an ABox graph constructed w.r.t. $canM$. Let $G_{\mathcal{T}} = (V_{\mathcal{T}}, E_{\mathcal{T}}, Ind_{\mathcal{T}}, Y_{\mathcal{T}})$ be a TBox graph constructed w.r.t. $canM$.

We start defining a relation $S_i \subseteq V \times V_i$, for $i \in \{0, 1, ..., |V_{\mathcal{A}}|\}$, where $temp_i = (V_i, E_i, Ind_i, Y_i)$ is the canonical graph of $mscG$ w.r.t. $canM$ that is constructed at each recursive call. The relation S_i is defined by following the steps of the Algorithm 1:

1. As shown in Line 4, the graph $mscG$ is initialized as an empty graph. Then the root node v_0 in $canM$ is added to $mscG_0$ together with its set of concepts. Therefore, $S_0 = \{(v_0, v_0')\}$, where v_0, v_0' are the root nodes of $canM$ and $mscG_0$ respectively.

2. Then, all nodes in $V_{\mathcal{A}}$ are visited only once, starting from the root node v_0 that contains the individual a (as shown in Algorithm 2):

- At each recursive call, a node v is the current node, $temp_i = (V_i, E_i, Ind_i, Y_i)$ is constructed to be the canonical graph of $mscG_i$ w.r.t. $canM$, where $mscG_i$ is the graph that is obtained at the recursive call i.
- Given a node v, a set $pe(E, v) = \{(v, r, v') \in E\}$ is the set of edges from $canM$ that contains v as a predecessor, a set $pe(E_i, v) = \{(v, r, v') \in E_i\}$ is the set of edges from $temp$ that contains v as a predecessor.
- An edge $e \in pe(E, v)$ is added to $mscG_i$ if there exists no edge $e' \in pe(E_i, v)$ such that e is simulated by e' (see Line 10 of Algorithm 2).
  An edge $e = (v, r, v') \in E$ is said to be simulated by an edge $e' = (d, r, d') \in E_i$ iff the following hold:
  - $Y(v) \subseteq Y_i(d)$ and $Y(v') \subseteq Y_i(d')$.
  - There exists a simulation S such that $(v', d') \in S$.
- Then we have one of two cases:
  (A) There exists an edge $e' = (v, r, d') \in pe(E_i, v)$ that simulates the edge $e = (v, r, v') \in pe(E, v)$ and $e' \in V_{\mathcal{T}}$, then there exists a simulation S such that $(v', d') \in S$. We then have that $S_i = S_{i-1} \cup S$.
  (B) Otherwise, $e = (v, r, v') \in pe(E, v)$ is added to $mscG_i$. Hence, $S_i = S_{i-1} \cup \{(v', v')\}$.

3. Algorithm 2 terminates after all nodes are visited and for every edge $e \in E$, there is an edge $e' \in E_i$ such that e is simulated by e'.

As a consequence of Lemma 4.2.10, we have that $temp_i \subseteq temp_{i+1}$, for all $0 \leq i < |V_{\mathcal{A}}|$. Therefore, if there exists a simulation from a node $v \in V$ to a node $v' \in temp_i$, then there exists also a simulation from $v \in V$ to $v' \in temp_{i+1}$, for all $0 \leq i < |V_{\mathcal{A}}|$. Let $temp_t = (V_t, E_t, Ind_t, Y_t)$ be the canonical graph of $mscG$ w.r.t. $canM$ that is constructed after the Algorithm 2 terminates.
Claim: For every $i \in \{0, ..., |V_{\mathcal{A}}|\}$, S_i satisfies:

(S'1) For every $(v, v') \in S_i$, $Y(v) \subseteq Y_t(v')$.

(S'2) If $i > 0$: for every $(v, v') \in S_{i-1}$ and $(v, r, e) \in E$, there exists $(v', r, e') \in E_t$, $(e, e') \in S_i$.

Proof of Claim: We prove by induction over i.
Base Induction: $I = 0$. Then, $S_0 = (v_0, v_0)$, where v_0, v_0 are the root nodes of $canM$ and $temp_t$. The Algorithm 1 copies v_0 to $mscG$ together with its concepts. Therefore, (A) is satisfied. (B) is also satisfied, because $i = 0$.
Induction Step: Assume $i > 0$ and that claim holds for S_{i-1}.

(S'1) Take some $(v, v') \in S_i$. If $(v, v') \in S_{i-1}$, then (S'1) holds from the induction hypothesis. Assume $(v, v') \in S_i \S_{i-1}$. This means we are in one of two cases:
  (I) From (A) we have that $S_i = S_{i-1} \cup S$ and $(v, v') \in S$, or
  (II) From (B) we have that $S_i = S_{i-1} \cup \{(v, v)\}$, and $Y(v) = Y_t(v)$.
  In case (I), S is a simulation, and thus $Y(v) \subseteq Y_t(v')$ by (S'1). In case (II), $Y(v) = Y_t(v)$ follows directly.

(S'2) Take some $(v, v') \in S_{i-1}$. If $(v, v') \in S_{i-2}$, then (S'2) holds from the induction hypothesis. Assume $(v, v') \in S_{i-1} \S_{i-2}$, and there exists an edge $(v, r, d) \in E$. Then there are two cases depending on whether $(v, r, d) \in E$ is simulated by some $(v', r, d') \in E_t$ or not:
  (I) From (A) we have that $S_i = S_{i-1} \cup S$ where $(d, d') \in S$ and $(v', r, d') \in E_t$. Or
  (II) From (B) we have that $S_i = S_{i-1} \cup \{(d, d)\}$, and $(v', r, d) \in E_t$.
  In both cases, we have that (S'2) is satisfied.

■

We write S_t to denote the relation $S_t \subseteq V \times V_t$, where $t = |V_{\mathcal{A}}|$.

From the claim we have that the relation $S_t \subseteq V \times V_t$ defines a simulation from $canM = (V, E, Ind, Y)$ to $temp_t = (V_t, E_t, Ind_t, Y_t)$ as follows:

- Take $(v, v') \in S_t \backslash S_{t-1}, (v, r, e) \in E$. Then e must be from S_t because we already processed all successors of nodes. We have one of two cases.
  - (I) e was not processed in the last step but in an earlier step. That means $S_i \subseteq S_{t-1}$, where $(e, e') \in S_i$. The Algorithm 2 would add (v', r, e') to E_{mscG} and $(e, e') \in S_i$, unless it is already there. Since this is the last step of the Algorithm 2, then (v', r, e') must be already in E_{mscG} and thus also in E_t. It follows that (S2) is also satisfied for (v, v').
  - (II) e was processed in the last step. Because we always process one node in each step, then $e = v$. This means that $(v, r, v) \in E$. If $(v', r, e') \in E_t$ such that $(canM, v) \lesssim (temp_{t-1}, e')$ through the simulation S, then $S_t = S_{t-1} \cup S$ and thus $(v, e') \in S_t$. Then also (S2) is satisfied for (v, v'). Otherwise, $E_t = E_{t-1} \cup \{(v', r, v)\}$ and $S_t = S_{t-1} \cup \{(v', v)\}$ so that (S2) is also satisfied.

Hence, the relation $S_t \subseteq V \times V_t$ defines a simulation from $canM$ to $temp_t$, where $temp_t$ is the canonical graph of $mscG$ w.r.t. $canM$ that is obtained after Algorithm 2 terminates. □

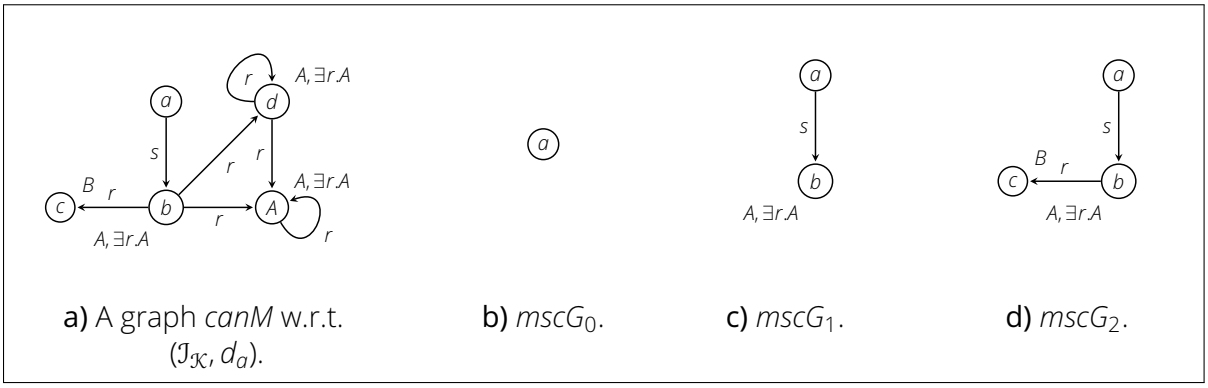
Lemma 4.4.2. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, graph $canM = (V, E, Ind, Y)$ be the graph of $(\mathfrak{I}_{\mathcal{K}}, d_a)$, and graph $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ be the graph obtained after the termination of Algorithm 1. It holds that $(canM, v_0) \lesssim (temp_t, v_0)$, where $temp_t = (V_t, E_t, Ind_t, Y_t)$ is the canonical graph of mscG w.r.t. canM.*

Proof. Again, after the recursive call of the Algorithm 2, The *mscG* graph is obtained such that the *mscG* graph is a sub-graph *canM*, and also $\mathfrak{I}_{mscG} \subseteq \mathfrak{I}_{canM}$ (recall the Definition 4.2.5). As consequence also the canonical graph $temp = (V_t, E_t, Ind_t, Y_t)$ of *mscG* w.r.t. *canM* is a sub-graph of the *canM* graph. Hence, also $\mathfrak{I}_{temp_t} \subseteq \mathfrak{I}_{canM}$. Therefore, there exists a simulation from *temp* to *canM*, due to that each node $v \in V_t$ has a corresponding node $v' \in V$ with $Y_t(v) \subseteq Y(v')$ and each edge $(v_1, r, v_2) \in E_t$ has a corresponding edge $(v_1', r, v_2') \in E$ with $Y_t(v_1) \subseteq Y(v_1')$ and $Y_t(v_1) \subseteq Y(v_2')$. Hence, it holds that $(temp, v_0) \lesssim (canM, v_0)$ after the Algorithm 1 terminates. □

Since there exists always a simulation equivalence between $(temp_t, v_0)$ and $(canM, v_0)$ after Algorithm 1 terminates and the fact that *mscG* contains only nodes from $V_{\mathcal{A}}$, it is not a sufficient condition to decide the existence of the msc w.r.t. a. It is still essential to check whether *mscG* contains a cycle or not. If there exists a cyclic path σ in *mscG*, then σ is not simulated by any path from $temp_t$. This allows us to obtain that if *mscG* is cyclic, then this cyclic path is generated by $\mathcal{A}$. Therefore, there exists no msc w.r.t. a. The reason is that $\mathfrak{I}$ that is obtained from *mscG* would contain a loop. Therefore, the tree unravelling of $\mathfrak{I}$ is infinite. One can employ the least tree unravelling of $\mathfrak{I}$ to construct the characteristic concept C. Then we can check whether C is the msc or not by Lemma 3.3.2.

Example 7. Consider the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$:

- $\mathcal{T} = \{A \sqsubseteq \exists r.A\}$.
- $\mathcal{A} = \{s(a, b), r(b, c), r(b, d), r(d, d), A(d), A(b), B(c)\}$.



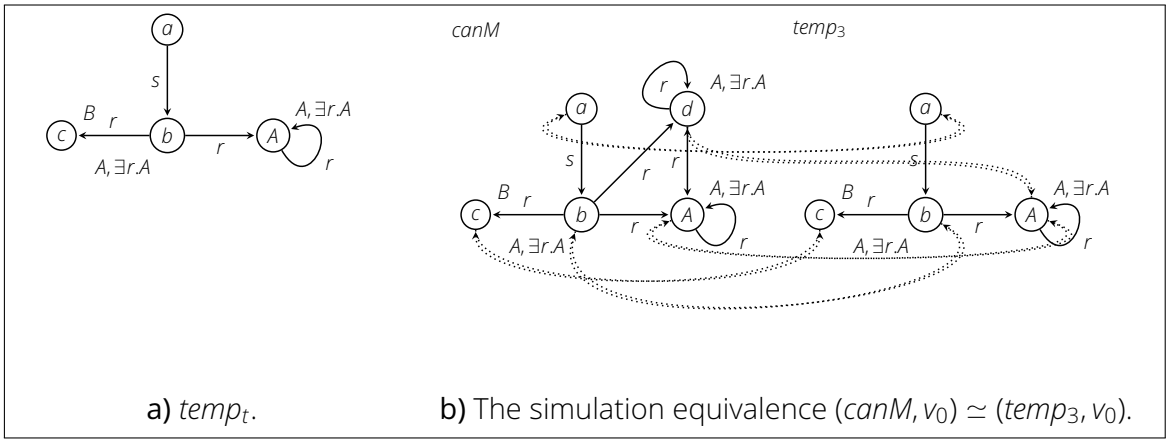
a) A graph *canM* w.r.t. $(\mathfrak{I}_\mathcal{K}, d_a)$. b) $mscG_0$. c) $mscG_1$. d) $mscG_2$.

Figure 4.5 *canM* constructed w.r.t. $(\mathfrak{I}_\mathcal{K}, d_a)$, the graphs $mscG_0, mscG_1, mscG_2$ obtained at the end of each recursive call.

Algorithm 2 starts with two initial graphs $mscG_0$ and *canM*. $mscG_0$ is initialized with only root node a in Algorithm 1. The graph $mscG_0$ contains only a and no edges. Then Algorithm 2 visits node a, and builds $temp_0$ to be the canonical graph of $mscG_0$ w.r.t. *canM*. Then it checks every edges $e \in pe(E, a)$ in *canM* whether there exists an edge $e' \in pe(E, a)$ in $temp_0$ such that e is simulated by e'. Since the node a contains only one edge $e = (a, s, b)$ where $Ind(a), Ind(b) \in N_{I,\mathcal{A}}$, therefore there is no simulation to it by any edge e from $temp_0$. Hence, the edge $e = (a, s, b)$ is added to $mscG_1$.

Then Algorithm 2 visits the next node which is b and builds $temp_1$ of $mscG_2$ w.r.t. *canM*. Since E_b in *canM* contains three edges $e_1 = (b, r, d), e_2 = (b, r, c), e_3 = (b, r, b)$, it checks whether any of these edges can be simulated by an edge from E_b in $temp_1$. In this case we have that $e_2 = (b, r, d)$ in E_b of *canM* is simulated by $e' = (b, r, A)$ in E_b of $temp_1$. Therefore, the Algorithm skips adding this edge to $mscG_1$. We have also that $e_3 = (b, r, b)$ in E_b of *canM* is simulated by $e' = (b, r, A)$ in E_b of $temp_1$. Hence, it is also skipped. However, the edge $e_1 = (b, r, c)$ is not simulated by any edge from E_b in $temp_1$. Hence, e_1 is added to $mscG_2$. Now we have only node c in *mscG* that should be visited before the recursive call terminates. In the last recursive call since the node c has no successors, the algorithm 2 terminates. In Figure 4.5, we can see the graphs generated of Example 7 by Algorithm 1.

Finally, there is still needed to check whether there exists a simulation equivalence between $(temp_t, v_0)$ and $(canM, v_0)$. We also need to check whether *mscG* contains a cycle. This two checks are done after the recursive call of Algorithm 2 and it returns to continue with remaining steps in Algorithm 1(see Figure 4.6).



a) $temp_t$. b) The simulation equivalence $(canM, v_0) \simeq (temp_3, v_0)$.

Figure 4.6 $temp_t$ constructed w.r.t. $mscG_2$ and *canM*, the simulation equivalence between $(temp_t, v_0)$ and $(canM, v_0)$.

As we can see in Figure 4.6 that after Algorithm 1 terminates, there exists a simulation equivalence between $(temp_t, v_0)$ and $(canM, v_0)$.

However, there could be the case that there exists the msc of a, even though *mscG* contains a cycle. To illustrate this consider the following example.

Example 8. Consider the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$:

- $\mathcal{T} = \{A \sqsubseteq \exists r.A\}$.
- $\mathcal{A} = \{r(a,b), r(b,a), r(a,c), r(c,d), A(d)\}$.

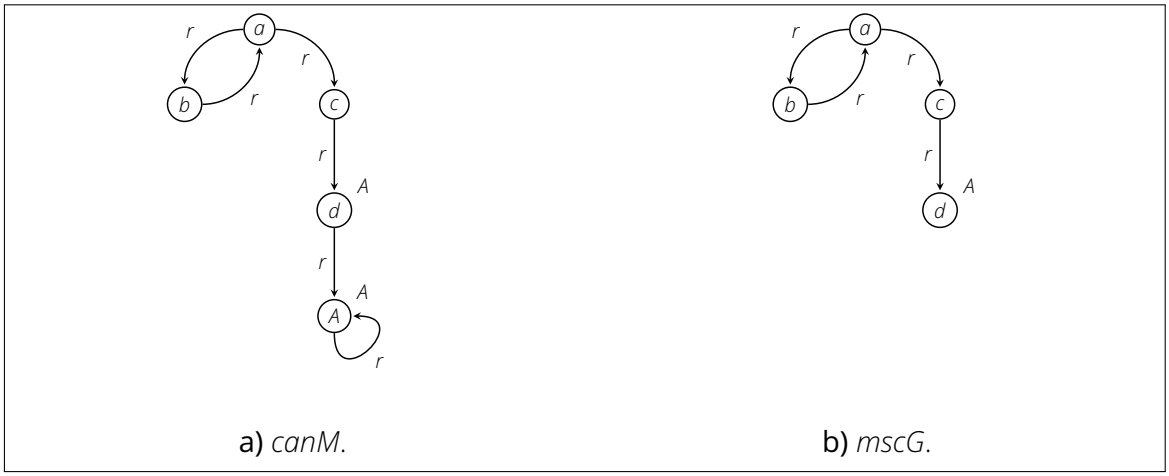


Figure 4.7 *canM* of $(\mathfrak{I}_\mathcal{K}, d_a)$ and *mscG* w.r.t. Example 8.

In the previous example, we have that *mscG* contains a cyclic path, although there exists the msc of a. The concept $C = \exists r.(\exists r.A)$ is the msc of a.

Therefore, it is not sufficient to check directly whether *mscG* is cyclic. It is still necessary to find the least sub-graph of *mscG* such that every path $\sigma = vr_1v_1r_2v_2\ldots$ in *mscG* is not simulated by any path $\sigma' = vr_1v_1'r_2v_2'\ldots$ in $temp_t$, with $v_i \neq v_i'$ for some $i > 0$.

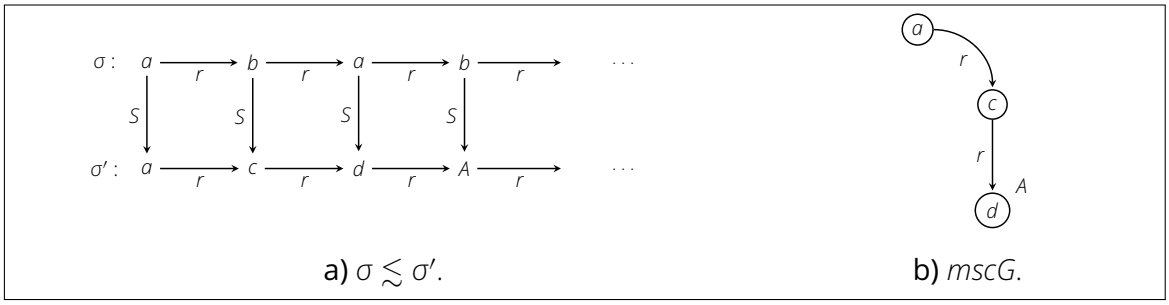


Figure 4.8 $\sigma \lesssim \sigma'$, where σ in *mscG* and σ' in $temp_t$, and the resulting sub-graph of *mscG*.

In Figure 4.8, we remove the path σ in *mscG* since it is simulated by the path σ. Hence, we obtain that the final *mscG* is the least graph such that $(temp_t, v_0)$ simulates $(canM, v_0)$. Then, if *mscG* is acyclic, then there exists the msc of a. One can construct the characteristic concept C w.r.t. $\mathfrak{I}$ of *mscG*.

Based on this construction of *mscG*, we observe the following.

Lemma 4.4.3. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, $canM = (V, E, Ind, Y)$ be the graph of $(\mathfrak{I}_\mathcal{K}, d_a)$, $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ be a graph constructed after the termination of Algorithm 1, and $temp_t = (V_t, E_t, Ind_t, Y_t)$ be the canonical graph of mscG w.r.t. canM. Then For every $(v, r, v_1), (v, r, v_2) \in E_t$ satisfy one of the following conditions:*

- $v_1 = v_2$.
- *Or,* $(temp_t, v_1) \not\lesssim (temp_t, v_2)$.

Proof. In Line 6 of Algorithm 1 it calls Algorithm 4, where Algorithm 4 intuitively finds a least graph of *mscG* such that the simulation equivalence relation between $(temp_t, v_0)$ and $(canM, v_0)$ still holds. By construction of the Algorithm 4 in Line 5, every edge $(v, r, v') \in E_{mscG}$ is removed from *mscG* if there exists an edge $(v, r, v'') \in E_t$ such that (v, r, v') is simulated by (v, r, v''). Therefore, Algorithm 1 ensures all edges $e \in E_{mscG}$ cannot be simulated by any edge $e' \in E_t$. This means given two edges $(v, r, v_1), (v, r, v_2) \in E_t$ we have one of the following cases:

(A) $(temp_t, v_1) \not\lesssim (temp_t, v_2)$.

(B) Or, $(temp_t, v_1) \lesssim (temp_t, v_2)$.

- Case (A) coincides with the our second condition.
- Case (B). Assume that $(temp_t, v_1) \lesssim (temp_t, v_2)$. Due to the fact that all edges $e = (v, r, v_1) \in E_{mscG}$ is not simulated by any edge $e' = (v, r, v_2) \in E_t$. Then, if $e = (v, r, v_1)$ is simulated by edge $e' = (v, r, v_2)$, then $e = e'$. Therefore, also $v_1 = v_2$.

□

As a consequence of the construction of Algorithm 1, one can observe that every edge $e = (v, r, v') \in E_{mscG}$ is not simulated by any edge $e' = (v, r, v'') \in E_t$ such that $v' \neq v''$, where $temp_t = (V_t, E_t, Ind_t, Y_t)$ is the canonical graph of *mscG* w.r.t. *canM* that is obtained after termination of the algorithm. This can be characterized by the following lemma.

Lemma 4.4.4. *Let* $canM = (V_{canM}, E_{canM}, Ind_{canM}, Y_{canM})$ *be the graph of* $(\mathfrak{I}_\mathcal{K}, d_a)$, $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ *the graph obtained after Algorithm 1 terminates,* $temp_t = (V_t, E_t, Ind_t, Y_t)$ *the canonical graph of mscG w.r.t. canM. Then, every infinite path* $\sigma = v_0 r_1 v_1 r_2 v_2 \ldots \in mscG$ *is not simulated by an infinite path* $\sigma' = v_0 r_1 v_1' r_2 v_2' \ldots \in temp_t$ *such that* $v_i' \in V_\mathfrak{T}$, $(mscG, v_i) \lesssim (temp_t, v_i')$, *and* $v_i \neq v_i'$ *for some* $i > 0$.

Proof. By construction of Algorithm 1:

1. As shown in Line 4, the graph *mscG* is initialized as an empty graph. Then the root node v_0 in *canM* is added to $mscG_0$ together with its set of concepts.

2. Then, all nodes in $V_\mathcal{A}$ are visited only once, starting from the root node v_0 that contains the individual *a* (as shown in Algorithm 2):
   - At each recursive call, a node *v* is the current node, $temp_i = (V_i, E_i, Ind_i, Y_i)$ is constructed to be the canonical graph of $mscG_i$ w.r.t. *canM*, where $mscG_i$ is the graph that is obtained at the recursive call *i*.
   - Given a node *v*, a set $pe(E, v) = \{(v, r, v') \in E\}$ is the set of edges from *canM* that has *v* as a predecessor, a set $pe(E_i, v) = \{(v, r, v') \in E_i\}$ is the set of edges from $temp_i$ that has *v* as a predecessor.
   - In Line 15 of Algorithm 2 an edge $e \in pe(E, v)$ is added to $mscG_i$ if there exists no edge $e' \in pe(E_i, v)$ such that *e* is simulated by e' (recall the Definition 4.3.4).

3. Let $temp_t = (V_t, E_t, Ind_t, Y_t)$ be the canonical graph of *mscG* w.r.t. *canM* after Algorithm 2 terminates.

4. Algorithm 2 terminates after all nodes are visited and for every edge $e \in E$, there is an edge $e' \in E_t$ such that *e* is simulated by e'. Also we can observe that every edge $e \in pe(E_{mscG}, v)$ is not simulated by any edge $e' \in pe(E_t, v) \wedge e' \neq e$, for every node $v \in V_{mscG}$. Otherwise, the edge *e* would not be added to *mscG*. Moreover, every edge $e = (v, r, v') \in E_{mscG}$ we have

that $v, v' \in V_{\mathcal{A}}$ ($V_{\mathcal{A}}$ is a set of nodes occurring in *canM* such that the nodes are generated by $\mathcal{A}$). Due to the fact that e is not simulated by any of the edges that are added from the canonical graph of *mscG* w.r.t. *canM* (recall the Definition 4.2.9).

Now Suppose that there exists a cyclic path in *mscG*. This means that there exists an infinite path $\sigma = v_0 r_1 v_1 r_2 v_2 ... \in mscG$ such that every edge $(v_{i-1}, r_1, v_i) \in E_{mscG}$ is not simulated by an edge $(v_{i-1}, r_1, v_i') \in E_t$, where $v_i' \in V_{\mathcal{T}}$ and $v_i \neq v_i'$, for all $i > 0$. Otherwise, the edge (v_{i-1}, r_1, v_i) would not exist in E_{mscG}. Hence, $\sigma = v_0 r_1 v_1 r_2 v_2 ... \in mscG$ is not simulated by any infinite path $\sigma' = v_0 r_1 v_1' r_2 v_2' ... \in temp_t$ such that $v_i' \in V_{\mathcal{T}}$ and $v_i \neq v_i'$ for some $i > 0$. □

We recall that Algorithm 1 constructs *mscG* graph such that $(temp_t, v_0) \simeq (canM, v_0)$. Also it checks whether *mscG* contains a cyclic path or not. If *mscG* contains a cyclic path, then there exists no msc of individual a. Otherwise, there exists an msc of a, and one then use convert *mscG* into an interpretation $\mathfrak{I}_{mscG}$ and use the interpretation to compute the msc.

Since deciding the existence of the msc with this approach does not rely on computing the concept, it is easy to see that it can be decided in polynomial time. Then, the constructed graph can be used to construct a possibly exponentially large concept. Also we can observe that role-depth of $(\mathfrak{I}_{mscG}, d_a)$ is bounded by $|V_{mscG}|$. Therefore, it is sufficient to construct up to $|V_{mscG}|$ to decide the existence of the msc.

4.5 proofs

4.5.1 Correctness

Theorem 4.5.1 (CORRECTNESS). *If the Algorithm 1 returns true, then there exists a most specific concept w.r.t. individual a.*

Proof. Assume that Algorithm 1 returns true. By the Definition 4.2.5, the interpretation $\mathfrak{I}_{canM}$ is constructed w.r.t. the *canM* graph, and the interpretation $\mathfrak{I}_{mscG}$ is constructed w.r.t. the *mscG* graph. A concept C is the characteristic concept obtained w.r.t. least tree unravelling of the interpretation $\mathfrak{I}_{mscG}$, where $k = |\Delta^{\mathfrak{I}_{mscG}}|$. This means that by Line 8, we have the following:

- There exists a *simulation equivalence* between $\mathfrak{I}_{canM}$ and the canonical model of the concept C w.r.t. $\mathcal{T}$ ($\mathfrak{I}_{canM} \simeq (\mathfrak{I}_{C,\mathcal{T}}, d_C)$.
- Graph *mscG* contains no cycle; Meaning that there is no path $v_1 e_1 ... e_n v_n$ with $v_n = v_1$, for all $e \in E$, and for all $v \in V$.

Since the *mscG* graph contains no cycle, then $\mathfrak{I}_{mscG}$ contains no cycle and by the definition of least tree unravelling, every path $\sigma = d r_1 d_1 ... r_n d_n$ of *mscG*, we have that $\{d, d_1, ..., d_n\}$ are distinct individuals; Tail(σ) $\notin \{d, d_1, ..., d_{n-1}\}$, where path $\sigma = d, d_1 r_1 ... r_n d_n$, for all paths $\sigma \in \mathfrak{I}'_{mscG_a}$. Moreover, By Lemmas 4.4.2 and 4.4.1 we have that $(temp, v_0) \simeq (canM, v_0)$, where *temp* is the canonical graph of *mscG* w.r.t. *canM*. While *mscG* is acyclic, it can be unravelled into a finite tree. Then the concept C is constructed w.r.t. the least tree unravelling of $\mathfrak{I}_{mscG}$. We have that $(\mathfrak{I}_{C,T}, d_C) \simeq (\mathfrak{I}_{\mathcal{K}}, d_a)$. By Lemma 3.3.2, we obtain that the concept C is the msc of the individual a. □

4.5.2 Completeness

Theorem 4.5.2 (COMPLETENESS). *If there exists a most specific concept w.r.t. individual a, then Algorithm 1 returns true.*

Proof. Assume that there exists a most specific concept w.r.t. individual a. Now suppose by contradiction that Algorithm 1 returns false. Then by Line 8, either:

(A) There exists no *simulation equivalence* between *canM* and the canonical graph *temp* of *mscG* w.r.t. *canM*, where *canM* is the graph of $(\mathfrak{I}_\mathcal{K}, d_a)$ and *mscG* is the graph obtained after the Algorithm 1 terminates.

(B) Or, the constructed $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ Graph has a cycle.

Case (A). Suppose that $(canM, v_0) \not\simeq (temp, v_0)$. Then it is a contradiction for Lemmas 4.4.2 and 4.4.1. After the Algorithm 1 terminates, there must exists a simulation equivalence between $(canM, v_0)$ and $(temp_t, v_0)$.

Case (B). Suppose that *mscG* is cyclic. By construction of Algorithm 1:

1. As shown in Line 4, the graph *mscG* is initialized as an empty graph. Then the root node v_0 in *canM* is added to $mscG_0$ together with its set of concepts.

2. Then, all nodes in $V_\mathcal{A}$ are visited only once, starting from the root node v_0 that contains the individual a (as shown in Algorithm 2):
   - At each recursive call, a node v is the current node, $temp_i = (V_i, E_i, Ind_i, Y_i)$ is constructed to be the canonical graph of $mscG_i$ w.r.t. *canM*, where $mscG_i$ is the graph that is obtained at the recursive call i.
   - Given a node v, a set $pe(E, v) = \{(v, r, v') \in E\}$ is the set of edges from *canM* that has v as a predecessor, a set $pe(E_i, v) = \{(v, r, v') \in E_i\}$ is the set of edges from $temp_i$ that has v as a predecessor.
   - In Line 15 of Algorithm 2, an edge $e \in pe(E, v)$ is added to $mscG_i$ if there exists no edge $e' \in pe(E_i, v)$ such that e is simulated by e' (recall the Definition 4.3.4).
   - If there exists an edge $e' = (v, r, d') \in pe(E_i, v)$ that simulates the edge $e = (v, r, v') \in pe(E, v)$, then we skip adding edge e to $mscG_i$. Otherwise, e is added to $mscG_i$.

3. Let $temp_t = (V_t, E_t, Ind_t, Y_t)$ be the canonical graph of *mscG* w.r.t. *canM* after Algorithm 2 terminates.

4. Algorithm 2 terminates after all nodes are visited and for every edge $e \in E$, there is an edge $e' \in E_t$ such that e is simulated by e'. Also we can observe that every edge $e \in pe(E_{mscG}, v)$ is not simulated by any edge $e' \in pe(E_t, v) \wedge e' \neq e$, for every node $v \in V_{mscG}$. Otherwise, the edge e would not be added to *mscG*. Moreover, every edge $e = (v, r, v') \in E_{mscG}$ we have that $v, v' \in V_\mathcal{A}$ ($V_\mathcal{A}$ is a set of nodes occurring in *canM* such that the nodes are generated by $\mathcal{A}$). Due to the fact that e is not simulated by any of the edges that are added from the canonical graph of *mscG* w.r.t. *canM* (recall the Definition 4.2.9).

So, if there is a cyclic path $\sigma = v_0 r_1 v_1 r_2 v_2 \ldots$ in the *mscG* graph, then $\sigma = v_0 r_1 v_1 r_2 v_2 \ldots$ contains elements from $V_\mathcal{A}$. Furthermore, from 4.4.4 we have that this path σ is not simulated by any cyclic path $\sigma' = v_0 r_1 v_1' r_2 v_2' \ldots \in temp_t$ such that $v_i' \in V_\mathcal{T}$, $(mscG, v_i) \lesssim (temp_t, v_i')$, and $v_i \neq v_i'$ for some $i > 0$.

Now suppose that a concept C_0 is $msc_\mathcal{K}(a)$. Then by Lemma 3.3.2 we have that $(\mathfrak{I}_{C_0,\mathcal{T}}, d_{C_0}) \simeq (\mathfrak{I}_\mathcal{K}, d_a)$. Let $G_0 = (V_{G_0}, E_{G_0}, Ind_{G_0}, Y_{G_0})$ be a graph constructed w.r.t. $(\mathfrak{I}_{C_0,\mathcal{T}}, d_{C_0})$. Since $canM = (V, E, Ind, Y)$ is the graph of $(\mathfrak{I}_\mathcal{K}, d_a)$, we have that $(G_0, v_0) \simeq (canM, v_0)$.

Also by Lemmas 4.4.2 and 4.4.1 we have that $(canM, v_0) \simeq (temp_t, v_0)$, where $temp_t = (V_t, E_t, Ind_t, Y_t)$ is the canonical graph of *mscG* w.r.t. *canM*. Let S, S' be simulations such that:

i) $S : (temp_t, v_0) \lesssim (canM, v_0)$.

ii) $S' : (canM, v_0) \lesssim (temp_t, v_0)$.

Since $(temp_t, v_0) \simeq (canM, v_0)$ and $(canM, v_0) \simeq (G_0, v_0)$, we have that there exists a simulation equivalence between G_0 and $temp_t$ i.e.,$((G_0, v_0) \simeq (temp_t, v_0))$. Let S_1, S_2 be simulations such that:

1) $S_1 : (temp_t, v_0) \lesssim (G_0, v_0)$.

2) $S_2 : (G_0, v_0) \lesssim (temp_t, v_0)$.

Due to the fact that *mscG* is cyclic, then there must exists an infinite path $\sigma = v_0 r_1 v_1 r_2 v_2 ... \in mscG$. Hence, from 1) there must exists an infinite path $\sigma' = v_0 r_1 v'_1 r_2 v'_2 ... \in G_0$ such that $(v_i, v'_i) \in S_1$ for all $i > 0$ and for some $j > 0$, $v'_j \in V_{\mathcal{T}}$ and $v'_{j-1} \notin V_{\mathcal{T}}$. Due to the fact that the concept C_0 can only be constructed from a finite tree unravelling. Therefore, the cyclic path comes only from the nodes in $V_{\mathcal{T}}$.

Moreover, from 2) there must exists a path $\sigma'' = v_0 r_1 v''_1 r_2 v''_2 ... \in temp_t$ such that such that $(v'_i, v''_i) \in S_2$ for all $i > 0$. We conclude that $\sigma \lesssim_{S_1} \sigma' \lesssim_{S_2} \sigma''$. Also, $(mscG, v_j) \lesssim_{S_1} (G_0, v'_j) \lesssim_{S_2} (temp_t, v''_j)$ for all $j > 0$.

By Lemma 4.4.4, we one of two cases:

A') $v_j = v''_j$.

B') Or, $v''_j \notin V_{\mathcal{T}}$ and $v_j \neq v''_j$.

Case A'). Assume that $v_j = v''_j$. Since $(v'_{j-1}, v''_{j-1}) \in S_2$, we have that $Y_{G_0}(v'_{j-1}) \subseteq Y_t(v''_{j-1})$. By Lemma 4.4.3 we have that v''_{j-1} has v'_j as a successor. Assume that $v''_j = v'_j \in V_{\mathcal{T}}$. Then, we have that $v_j = v''_j = v'_j$. We have that the cyclic path σ in *mscG* contains only nodes v such that $v \in V_{\mathcal{A}}$. Then it is a contradiction that $v_j = v''_j = v'_j$, where $v'_j \in V_{\mathcal{T}}$.

Case B'). Assume that $v''_j \notin V_{\mathcal{T}}$ and $v_j \neq v''_j$. Then, by Lemma 4.4.4 we have either one of the following conditions:

i') $v_j = v''_j$.

ii') Or, $(mscG, v_j) \not\lesssim (temp_t, v''_j)$.

- Case i'). Suppose that $v_j = v''_j$. Then, it is a contradiction for the assumption that $v_j \neq v''_j$.
- Case ii'). Suppose that $(mscG, v_j) \not\lesssim (temp_t, v''_j)$. Then it also contradicts the assumption that $\sigma \lesssim \sigma''$. Due to the fact that $(mscG, v_j) \lesssim (temp_t, v''_j)$ for all $j > 0$.
  Therefore, it is a contradiction for Case B').

Hence, we have that a contradiction for Case (B) that *mscG* is acyclic. Since we have a contradiction for Cases (A) and (B), we conclude that if there exists a most specific concept w.r.t. individual a, then Algorithm 1 must return true. □

4.5.3 Polynomial Time Complexity

Theorem 4.5.3 (COMPLEXITY). *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base. the Algorithm 1 decides the existence of the most specific concept w.r.t. an individual a in polynomial time.*

Proof. We start with Algorithm 1, which constructs the $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ graph from $canM = (V, E, Ind, Y)$. Starting from the root node in *canM*, Algorithm 2 is called recursively, to visit nodes $v_1 \in V$ and adds edges $e_1 \in E$ to E_{mscG}. At each recursive call, The $temp_i = (V_i, E_i, Ind_i, Y_i)$ graph is the canonical graph of *mscG* w.r.t. *canM* with i is the number of recursive calls of the Algorithm 2.

Then we go over all edges $e_1 = (v_1, r, v'_1) \in E \wedge v_1 \in V$, and check whether there exists an edge $e_2 = (v_2, r, v'_2) \in E_i \wedge v_2 \in V_i$, such that e_1 is simulated by e_2. It is known that constructing such a simulation between two graphs can be computed in polynomial time in the size of two graphs *canM* and $temp_i$ [HHK95]. Therefore, Algorithm 2 can be computed in polynomial time w.r.t. the size of two graphs $\mathcal{O}((|V| \times |E|)^3)$, given that the simulation has the complexity of $\mathcal{O}(|V| \times |E|)$.

Assuming that the *canM* graph is a Complete graph, where all vertices are connected directly with an edge, then $|E| = |V|^2$. The worst-case time complexity will be $\mathcal{O}(|V|^7)$. Assuming that the *canM* graph is a path; Meaning that each node has only one successor, then $|E| = 1$. The best-case time complexity will be $\mathcal{O}(|V|)$. Since both best and worst-case time complexities belong to the polynomial time complexity class, the Algorithm 1 decides the existence of the most specific concept w.r.t. the individual a in polynomial time. □

5 Implementation

In this chapter, we specify an algorithm for deciding the existence of the msc w.r.t. an individual, and it computes the concept if it exists. Moreover, we present an example to describe the construction of the msc.

5.1 The Algorithm

In this section, we describe all the algorithms that we were using to compute the msc if it exists. We use a popular reasoner to reason about ontology. ELK [KKS12] is a specialized reasoner for the lightweight ontology language OWL $\mathcal{EL}$. It is used to compute the concept related to each individual. Also, it is used in building the canonical model of $\mathcal{A}$. The program is available at [Nad], where Java programming [AGH05] is used. The reason to choose the programming language is that the OWL API [HB09] is available in Java. OWL API is a Java API that allows us to create and manipulate OWL ontologies.

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, a be an individual. Since we assume that $\mathcal{T}$ is normalized TBox, then we normalize $\mathcal{T}$. Then, we check whether the individual a exists in $\mathcal{A}$. If $a \notin N_{I,\mathcal{A}}$, then return false. Otherwise, we build the canonical model $\mathfrak{I}_\mathcal{K}$ of $\mathcal{K}$. Then we construct the pointed interpretation $(\mathfrak{I}_\mathcal{K}, d_a)$ from $\mathfrak{I}_\mathcal{K}$. Now the interpretation $(\mathfrak{I}_\mathcal{K}, d_a)$ are used as inputs for the Algorithm 1.

The Algorithm 1 starts with constructing a graph $canM = (V, E, Ind, Y)$ from $(\mathfrak{I}_\mathcal{K}, d_a)$ (See Line 1 of Algorithm 1). Then initializes a graph $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ as an empty graph. Furthermore, the root node containing the individual a is added to $mscG$. The algorithm traverses all nodes of $canM$ and adds edges and nodes to $mscG$ if needed. For this reason, the algorithm uses two lists *visited* and *non-visited* to keep track of the visited and the non-visited nodes in $canM$. Recursively visit nodes of $canM$ starting from the root node. This recursive call is done using the Algorithm 2.

The Algorithm 2 takes initially the root node v, two graphs $canM$, $mscG$ and the lists *visited*, *non-visited* as inputs. It checks whether the node v is already visited before or not. If yes, then it terminates. Otherwise, it adds the node v to *visited* to make sure that the algorithm will terminate after visiting all nodes only once. Then it builds a graph $temp_i = (V_i, E_i, Ind_i, Y_i)$ to be the canonical graph of $mscG$ w.r.t. $canM$ (recall the Definition 4.2.9), where i is the number of the recursive calls of Algorithm 2. Let $pe(E, v)$ be the set of edges in $canM$ with the node v as predecessor, and $pe(E_i, v)$ be the set of edges in $mscG$ with the node v as predecessor. It goes over the edges of $pe(E, v)$ and check for every edge $e = (v, r, v') \in pe(E_1, v)$ whether there exists an edge $e' = (v, r, d) \in pe(E, v)$ such that e is simulated by e' (recall the Definition 4.3.4). If no, then the algorithm adds e to $mscG$ with its nodes. Furthermore, it checks whether the

successor node v' has been visited before or not. If yes, then it is ignored. Otherwise, it is added to the *non-visited* list. After checking every edge $e \in pe(E, v)$ and adding all edges that are not simulated by any edge $e' \in pe(E_i, v)$, the algorithm picks a new node to be visited in its next recursive call. For that, the algorithm picks the first element in the *non-visited* list. Picking the first element of the list ensures that the nodes of *canM* are traversed in the breadth-first order.

Algorithm 3 is the algorithm that checks given two edges e, e' whether there exists a simulation from e to e'. It takes an edge $e = (v_1, r, v_1')$ from *canM* and an edge $e' = (v_2, r, v_2')$ from $temp_i$. Then it checks whether $Y(v_1) \subseteq Y_i(v_2)$ and $Y(v_1') \subseteq Y_i(v_2')$. If yes, then it checks whether there exists a simulation from *canM* to *temp* starting from v_1' in *canM* and v_2' in $temp_i$. If it also returns TRUE, then e' simulates e and returns TRUE. If any of these checks fail, then e' does not simulate e, and it will return FALSE.

After the recursive call of the Algorithm 2, it is still essential to construct the least graph of *mscG*, where every edge e in E_{mscG} is not simulated by any edge e' in E_{temp_t}. Algorithm 4 is the algorithm which performs this operation. It generates all edges of nodes recursively starting from the root node v_0 and does the following:

- The set (E_{mscG}, v_i) is the set containing all edges e of *mscG* with root node v_i. The set (E_{temp_t}, v_i) is the set containing all edges e of *temp* with root node v_i.
- for each edge $e \in (E_{mscG}, v_i)$, we check whether there exists a edge $e' \in (E_{temp_t}, v_i)$ such that $e \lesssim e'$ and $e \neq e'$.
- If the edge e is simulated by e', then we remove the edge e from E_{mscG}.

After Algorithm 4 removes all simulated edges, we still need to remove the nodes $v \in V_{mscG}$ such that v is not reachable from v_0. Then, we obtain the least graph *mscG* such that $(temp_t, v_0) \simeq (canM, v_0)$.

Finally, the Algorithm 1 checks whether there exists a simulation equivalence between $(canM, v_0)$ and $(temp_t, v_0)$, where $temp_t$ is the canonical graph of *mscG* w.r.t. *canM* that is obtained after Algorithm 2 terminates. Also, it checks whether *mscG* contains a cycle. If both checks return TRUE, then Algorithm 1 returns TRUE and *mscG* is the most specific concept w.r.t. the individual a. Otherwise, there is no msc w.r.t. a.

Algorithm 1: BuildMsc function. For an individual a and an ontology O. The function call is BuildMsc(a,O)

Input: a, O `/* individual a and ontology O */`
Output: Boolean

1 $(V, E, Ind, Y) \leftarrow CanonicalModel(a)$ `/* set` $(\mathfrak{I}_{\mathcal{K}}, d_a)$ `to canM */`
2 $non\text{-}visited \leftarrow \{v_0 \in V \mid v_0 \text{ is root node}\}$ `/* initialize the set of non visited nodes to root node` v_0 `of canM */`
3 $visited \leftarrow \emptyset$ `/* initialize the set of visited nodes to empty */`
4 $(V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG}) \leftarrow (\{v_0 \in V\}, \emptyset, \langle v_0; Y(v_0)\rangle)$ `/* intialize mscG with` $v_0 \in V$ `and its concepts */`
5 $visitNode(non\text{-}visited.pop(), non\text{-}visited, visited, mscG, canM)$ `/* recursively add nodes and edges from canM to mscG */`
6 $LeastGraph(mscG, canM)$ `/* construct the least graph of` *mscG* `*/`
7 $(V_t, E_t, Ind_t, Y_t) \leftarrow CanonicalGraph(mscG, canM)$ `/* construct the canonical graph` $temp_t$ `*/`
8 **if** $(temp_t, v_0) \simeq (canM, v_0)$ *AND* $Cyclic(mscG) = FALSE$ `/* check whether mscG is cyclic and there exists a simulation equivalence between` $(temp_t, v_0)$ `and` $(canM, v_0)$ `*/`
9 **then**
10 `  ` **return** TRUE `/* Msc exists */`
11 **else**
12 `  ` **return** FALSE `/* No msc exists */`

Algorithm 2: visitNode recursive function. For a node v, a set *non-visited* of non visited nodes, a set *visited* of visited nodes, a graph $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$ and graph $canM = (V, E, Ind, Y)$. The initial function call is visitNode(v_0,non-visited,visited,mscG,canM).

```
Input: v, non-visited, visited, mscG, canM  /* node v, sets non-visited and visited of visited
       and non visited nodes, Graphs mscG and canM */
Output: mscG                                /* the constructed mscG */
1  if v ∉ visited                           /* check whether v is not visited before */
2  then
3      (V_temp, E_temp, Ind_temp, Y_temp) ← CanonicalGraph(mscG, canM) /* construct temp_i of mscG_i
          w.r.t. canM                                                  */
4      visited ← visited ∪ {v ∈ V}          /* add v to set visited */
5      for (v, r, d) ∈ pe(E, v) ∧ d ∈ V_A   /* loop over the edges e ∈ pe(E, v), and successor node d
        is an individual */
6      do
7          simulated ← FALSE                /* flag for the simulation of edges */
8          for (v, r, d') ∈ pe(E_temp, v)   /* find a corresponding edge e' ∈ pe(E_temp_i, v) such that e ≲ e'
             */
9          do
10             if simulatedBy((v, r, d), (v, r, d'), canM, temp) = TRUE   /* check whether
                (v, r, d) ≲ (v, r, d') */
11             then
12                 simulated ← TRUE         /* found a simulation from (v, r, d) to (v, r, d') */
13         if simulated = FALSE             /* check whether there is no simulation with any edge
            e' ∈ pe(E_temp_i, v) to e ∈ pe(E, v) */
14         then
15             mscG ← (V_mscG ∪ d ∈ V, E_mscG ∪ (v, r, d) ∈ pe(E, v), Y_mscG ∪ ⟨d; Y(d)⟩)   /* add edge
                (v, r, d) ∈ pe(E, v) and node d ∈ V to mscG */
16             if d ∉ visited               /* check whether node d is already visited before */
17             then
18                 non-visited ← non-visited ∪ {d ∈ V}  /* add node d to the non visited set of
                    nodes */
19 else
20     if non-visited ≠ ∅                   /* non-visited contains nodes that still need to be visited */
21     then
22         visitNode(non-visited.pop(), non-visited, visited, mscG, canM)   /* visit the first
            element of non-visited */
23 return mscG                              /* return the constructed graph mscG */
```

Algorithm 3: SimulatedBy function. For an edge $(v, r, d) \in pe(E, v)$, an edge $(v, r, d') \in pe(E_{temp}, v)$, a graph $canM = (V, E, Ind, Y)$ and $temp = (V_{temp}, E_{temp}, Ind_{temp}, Y_{temp})$. The function call is $SimulatedBy((v, r, d), (v, r, d'), canM, temp)$

```
Input: (v,r,d), (v, r, d'), canM, temp        /* edges (v,r,d), (v,r,d') and graphs canM, temp */
Output: Boolean
1 if Y(d) ⊆ Y_temp(d')        /* check whether the set of concepts on d is subset of the set of
    concepts on d' */
2  then
3      if (canM, d) ≲ (temp, d')        /* check whether there exists a simulation from (canM,d) to
         (temp,d') */
4      then
5          return TRUE                                   /* (v,r,d) ≲ (v,r,d') */
6      else
7          return FALSE                                  /* (v,r,d) ⋦ (v,r,d') */
8  else
9      return FALSE                                      /* (v,r,d) ⋦ (v,r,d') */
```

Algorithm 4: LeastGraph function. For a graph $mscG = (V_{mscG}, E_{mscG}, Ind_{mscG}, Y_{mscG})$, a graph $canM = (V, E, Ind, Y)$. The function call is LeastGraph(mscG,canM)

```
Input: mscG, canM                                   /* graphs mscG and canM */
Output: mscG                                        /* the least graph of mscG */
1 for v ∈ V_mscG do
                                                    /* remove edges from mscG */
2     temp ← CanonicalGraph(mscG, canM)
3     for e ∈ (E_mscG, v) do
4         for e' ∈ (E_temp, v) do
5             if e ≲ e' AND e ≠ e' then
6                 E_mscG ← E_mscG \ {e}             /* remove the simulated edges from E_mscG */
7             '
8 for d ∈ V_mscG do
                                                    /* remove nodes from mscG */
9     if pe(E_mscG, d) = ∅                          /* check whether d has no successors */
10     then
11         if (v', r, d) ∉ E_mscG                   /* check whether d has no predessor */
12         then
13             V_mscG ← V_mscG \ {d ∈ V_mscG}       /* remove the node that is not reachable and has
                 no successors */
14 return mscG
```

6 Evaluation

In this chapter, we provide an experimental evaluation of a different set of benchmarks. We state the hardware setup that was used to evaluate the benchmarks. Then, we provide the selected set of instances from the different benchmarks. Finally, we provide experimental results to state the difference between the previous and the new approach within a timeout of 1,600 seconds.

6.1 Experimental Setting

6.1.1 Hardware Setup

All experiments have been performed on a machine equipped with 11th Gen Intel(R) Core(TM) i7-11370H with four 3.30GHz cores and 16GB of RAM operated by Windows 11 (21H2 version). Each task has been given a 1,600-second timeout. This timeout includes the time required to decide the existence of the msc and the time required to manipulate the input ontology.

6.1.2 Benchmarks

We considered a set of instances from three benchmark sets. The set consists of 155 instances from BioPortal which includes ontologies from applications in biology and medicine [Noy+09], 109 instances from OWL reasoner evaluation (ORE) 2015 competition [Par+17], and 2091 instances from the Manchester OWL repository [Mat+14]. This set is filtered to contain only those instances such that each instance contains at least one individual with at least one successor. We write 'ORE', 'BIO' and 'MOR' to express the instances from OWL reasoner evaluation (ORE) 2015 competition, BioPortal and the Manchester OWL repository respectively. We write 'ALL' to express the full set of instances. For each instance, we select the individual with the most number of successors as our input.

6.2 Experimental Results

In this section, we provide experimental results to compare the new approach with the previous approach. First, we show the number of decided and undecided instances w.r.t. a timeout of 1,600 seconds. Then, we show the time required to decide the existence of the msc. Finally, we show the difference between the maximal role-depth k that would have been used for Lemma 3.3.3, and the role-depth actually explored by our method.

In Table 6.1, the leftmost "Benchmark" column indicates each benchmark set, while "instance" indicates the number of instances. Sub-columns "Decided" and "Undecided" represent the number of solved and unsolved instances within the timeout of 1,600 seconds. The sub-column "Total" represents the total number of the selected instances per each set. The row "ALL" represents the full set of the selected instances.

Table 6.1 The number of decided and undecided instances with a 1,600 seconds timeout.

	Number of Instances		
Benchmark	Decided	Undecided	Total
ORE	95	14	109
BIO	148	7	155
MOR	2003	88	2091
ALL	2246	109	2355

We acknowledge that all experimental results and all drawn conclusions are based on a single benchmark set.

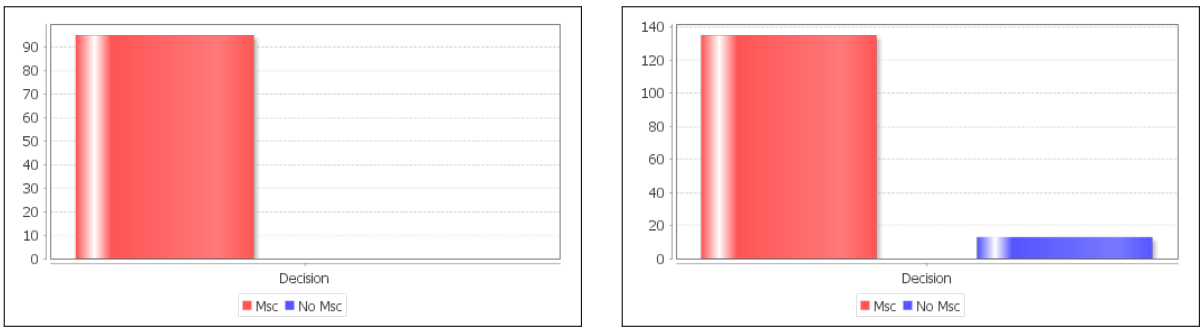
Table 6.2 The comparison between the maximal role-depth of the previous and new approach.

		Role-Depth			
Benchmark Set		min	median	mean	max
ORE	Previous	2315.00	61779602.00	344157725.99	2075238298.00
	New	1.00	1.00	1.67	10.00
BIO	Previous	3.00	995144.50	25039500.90	2101075370.00
	New	1.00	2.50	1.89	11.00
MOR	Previous	3.00	327.00	13954266.52	2082188164.00
	New	1.00	2.00	1.80	21.00
ALL	Previous	3.00	2250.50	28651480.83	2101075370.00
	New	1.00	1.50	1.80	21.00

In Table 6.2, the leftmost "Benchmark" column indicates each benchmark set, and "Role-depth" indicates the role-depth explored by Algorithm 1 and respectively the role-depth bound from Lemma 3.3.3. Sub-columns "min", "median", "mean", and "max" present information about instances which did not include time out for a given setup in seconds. Their meaning is self-explanatory, i.e. "min" and "max" stand for minimal and maximal role-depth required to decide the existence of the msc in an instance within a setup. "Mean" and "median" denote the mean and median of role-depth required needed to solve all instances within a setup. As mentioned before, timed-out instances are not included in this analysis. Rows "Previous" and "New" indicate the computation of the role-depth in the previous- and the new approach per each benchmark set.

As we can see that the role-depth of the interpretation with the new approach is significantly smaller than the one we obtained from the previous one.

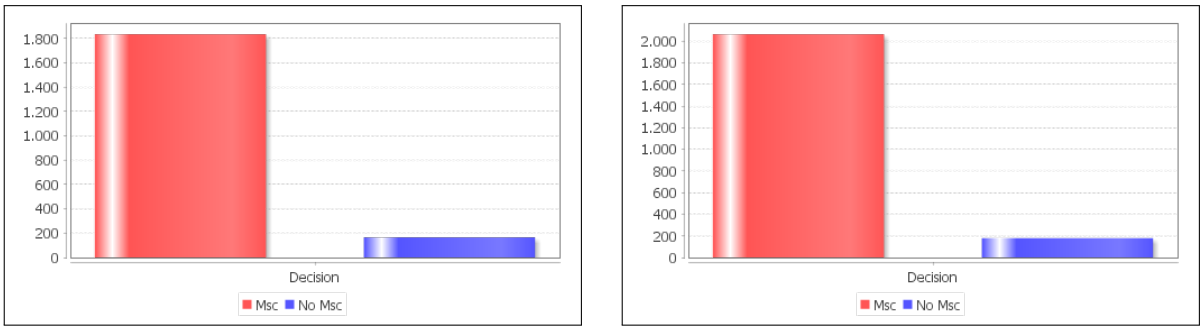
Figure 6.1 shows how often there existed an MSC, respectively did not exist one, w.r.t. the individual, for the benchmarks "ORE" and "BIO". It decides whether there exists or not within the timeout. Therefore, we exclude those instances that exceeded the timeout. In the "ORE" benchmark set of Figure 6.1a, we can see that Algorithm 1 was able to find the msc of an individual in all instances without exceeding the timeout. Therefore, the number of instances where the msc of an individual exists was 95. However, in the "BIO" benchmark set of Figure 6.1b,



a) ORE b) BIO

Figure 6.1 Bar plots indicating the number of instances with msc and No msc w.r.t. "ORE" and "BIO" within the timeout.

we have that the number of instances with no msc exists was 13, while the number of instances with the msc was 135.



a) MOR b) ALL

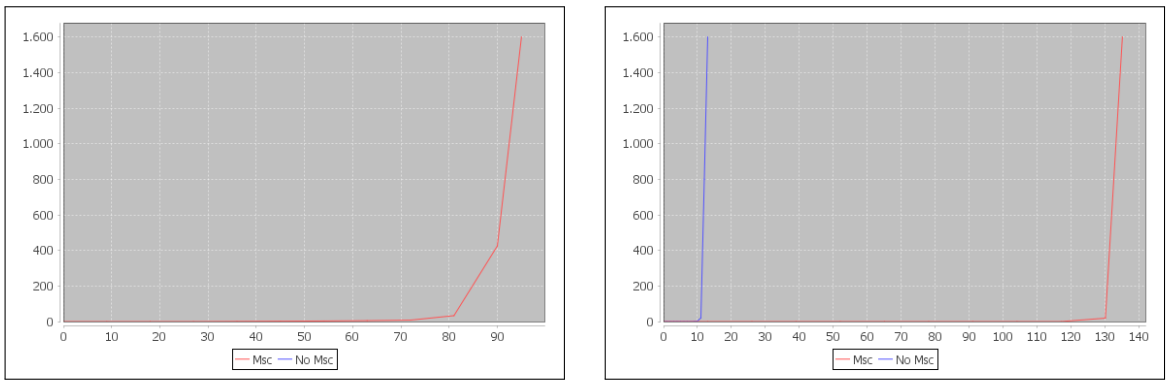
Figure 6.2 Bar plots indicating the number of instances with msc and No msc w.r.t. "MOR" and "ALL" within the timeout.

Figure 6.2 indicates the existence of the msc w.r.t. an individual for both "MOR" and "ALL" benchmarks. It decides whether there exists or not within the timeout. Therefore, we exclude those instances that exceeded the timeout. In the "MOR" benchmark set of Figure 6.2a, the number of instances with msc exists was 1837, while the number of instances with no msc exists was 166. Also, In the "ALL" benchmark set, we have that the number of instances with msc exists was 2067, while the number of instances with no msc exists was 179. Table 6.3 summarizes the number of instances with msc and no msc found that was computed with Algorithm 1.

Table 6.3 The number of instances with msc and with no msc exists.

	Number of Instances		
Benchmark	Msc	No Msc	Total
ORE	95	0	95
BIO	135	13	148
MOR	1837	166	2003
ALL	2067	179	2246

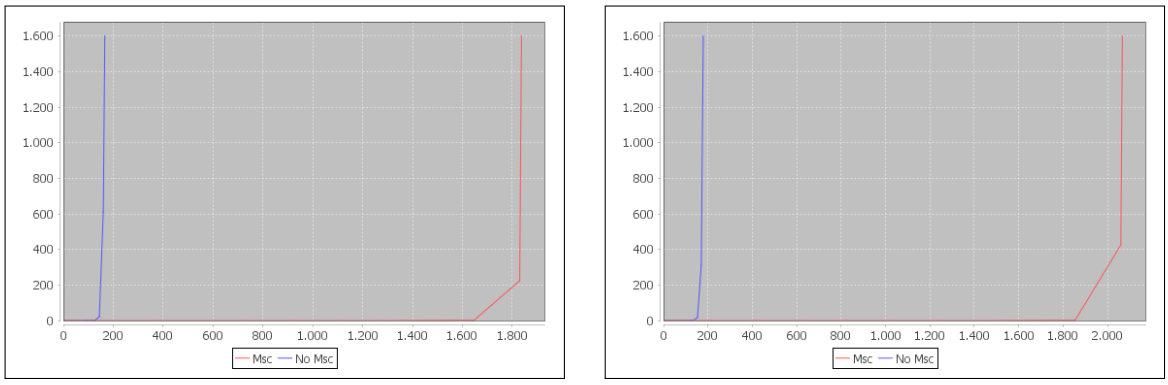
Figure 6.3 indicates the time required to decide the existence of the msc w.r.t. an individual



a) ORE b) BIO

Figure 6.3 Line plots indicating the time required to decide the existence of the msc. In both plots, the instances are sorted in ascending order by the time required, where x-axes indicate the time (in seconds) and y-axes indicate the number of instances.

within 1,600 seconds, where the line highlighted in "red" indicates the time for instances with the msc and the line highlighted in "blue" indicates the time for instances without the msc. Since all instances of "ORE" have the msc, therefore we can see in Figure 6.3a that there is no line highlighted "blue". In Figure 6.3b, we see that the number of instances without the msc required more time to be computed than the number of instances with the msc in "BIO" instances.

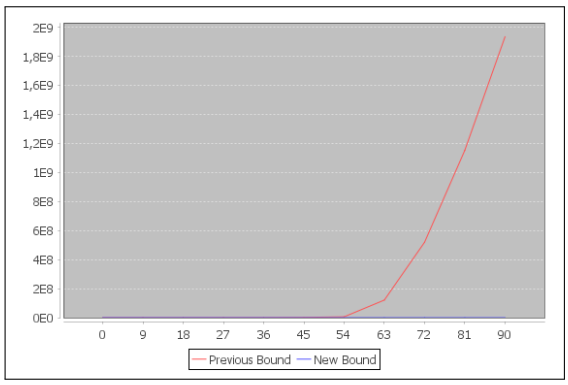


a) MOR b) ALL

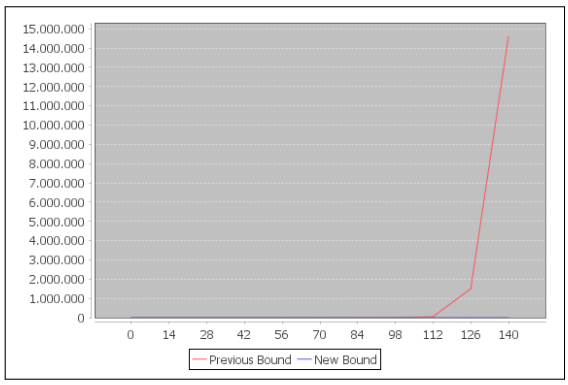
Figure 6.4 Line plots indicating the time required to decide the existence of the msc. In both plots, the instances are sorted in ascending order by the time required, where x-axes indicate the time (in seconds) and y-axes indicate the number of instances.

Figure 6.4 indicates the time required to decide the existence of the msc w.r.t. an individual within 1,600 seconds, where the line highlighted in "red" indicates the time for instances with the msc and the line highlighted in "blue" indicates the time for instances without the msc. Figure 6.4a of "MOR" instances shows that the number of instances without the msc exists is smaller than the number of instances with the msc. However, the time consumed in the instances with no msc w.r.t. an individual is larger than the time consumed in the instances with msc. This is due to the time required to find the least graph and remove the simulated paths in the cyclic graphs.

Figures 6.5 and 6.6 indicate the computed bounded role-depth of the previous- and the new approach, where the line highlighted in "red" indicates the role-depth of the previous approach



a) ORE

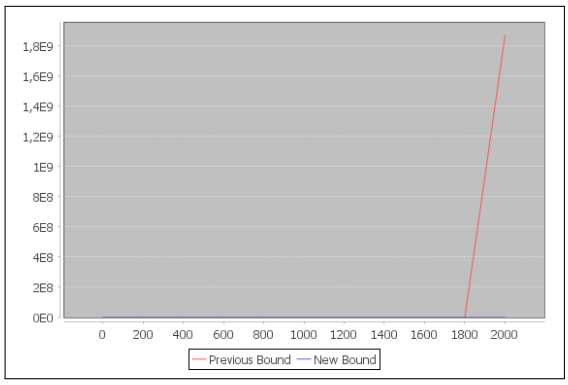


b) BIO

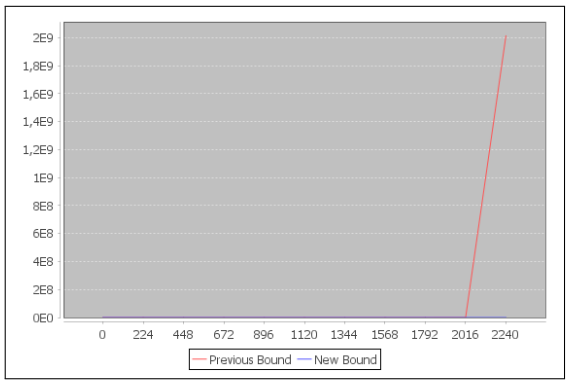
Figure 6.5 Line plots indicating the bounded role-depth of the previous- and new approach. In both plots, the instances are sorted in ascending order by the bounded role-depth, where x-axes indicate the bounded role-depth and y-axes indicate the number of instances.

and the line highlighted in “blue” indicates the role-depth of the new approach.

As we can see that the overall bounded role-depth of the new approach has a significantly smaller number than the one we obtained from the previous approach. Therefore, deciding the existence of the msc w.r.t. an individual using the new approach is much better in terms of performance and time required than deciding the existence of the msc w.r.t. an individual using the previous approach. Thus, constructing the concept w.r.t. the new bound is much smaller in terms of the size of the concept than constructing the concept w.r.t. the previous bound.



a) MOR



b) ALL

Figure 6.6 Line plots indicating the bounded role-depth of the previous- and new approach. In both plots, the instances are sorted in ascending order by the bounded role-depth, where x-axes indicate the bounded role-depth and y-axes indicate the number of instances.

We can summarize that the new approach provides an efficient mechanism for deciding the existence of the msc. Also, it has a smaller bound than the previous bound. Finally, the new approach does not rely on constructing the concept. Thus, it is polynomial even if constructing the concept could take exponential time in case that concept was exponentially large. Although computing the concept could take exponential time, it is still better to compute it with our new method. The reason is that the role-depth computed by our new method is significantly smaller than the one obtained from Lemma 3.3.3. Thus, computing the concept from the new approach would take less time than computing it from the previous approach even that the

concept was exponentially large.

7 Conclusion and Future Work

Computing the most specific concept is a non-standard inference task that supports bottom-up construction of the knowledge base. It is the least concept which has an individual as an instance.

We have revisited the previous approach from [Baa03a] which computes the most specific concept w.r.t. an individual in the presence of terminological TBoxes.

We have presented the previous approach for deciding the existence of the msc w.r.t. an individual from [ZT13] in the presence of general TBoxes. We have shown that in practice, the previous approach does not explain how to decide the existence of the msc practically without constructing the msc explicitly, which could be exponentially large.

In Chapter 4, we have introduced a new approach for deciding the existence of the msc w.r.t. an individual without relying on constructing the concept. The main idea of the new approach is to construct a sub-interpretation of the canonical model such that there exists a simulation equivalence between the canonical graph of the sub-interpretation and the canonical model. We observed that it is only necessary to construct the sub-interpretation in the cyclic ABoxes, as the msc always exists in the acyclic ABoxes.

Moreover, in Chapter 5, we have presented an implementation of our approach. We have introduced an efficient method to decide the existence of the msc practically. We have provided correctness and completeness proofs of our newly introduced approach. Also, we have shown that it has polynomial run-time.

Finally, in Chapter 6, we have provided an experimental evaluation to state the difference between both approaches in terms of bounded role-depth. Also, we have shown the time performance of the new approach for deciding the existence of the msc. We have observed that our new approach has smaller maximal role-depth than the one obtained from the previous approach from [ZT13]. Also, the concept computed with our new approach is smaller the one computed from the previous approach. We have acknowledged that the method takes a longer time in the instances the msc does not exist than the instance with msc exists.

7.1 Future Work

The Future work on the practical side is to implement the newly introduced approach in the context of ontology repairing. This can be achieved by integrating the new approach with the procedure provided by [Kri] and integrate it with the existing tools [ET12].

On the theoretical side, we would like extend the results towards knowledge bases formulated in more expressive Horn-DLs than $\mathcal{EL}$.

Bibliography

[AGH05] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[Baa+07] Franz Baader et al., eds. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2007.

[Baa+17a] Franz Baader et al. "An Introduction to Description Logic". In: 2017.

[Baa+17b] Franz Baader et al. *An Introduction to Description Logic*. Cambridge University Press, 2017. DOI: 10.1017/9781139025355.

[Baa+21] Franz Baader et al. "Computing Optimal Repairs of Quantified ABoxes w.r.t. Static EL TBoxes". In: *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 309–326. ISBN: 978-3-030-79875-8. DOI: 10.1007/978-3-030-79876-5_18. URL: https://doi.org/10.1007/978-3-030-79876-5_18.

[Baa+22] Franz Baader et al. "Optimal ABox Repair w.r.t. Static *EL* TBoxes: From Quantified ABoxes Back to ABoxes". In: *The Semantic Web - 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29 - June 2, 2022, Proceedings*. Ed. by Paul Groth et al. Vol. 13261. Lecture Notes in Computer Science. Springer, 2022, pp. 130–146. DOI: 10.1007/978-3-031-06981-9_8. URL: https://doi.org/10.1007/978-3-031-06981-9%5C_8.

[Baa03a] F. Baader. *The Instance Problem and the Most Specific Concept in the Description Logic $\mathcal{EL}$ w.r.t. Terminological Cycles with Descriptive Semantics*. LTCS-Report LTCS-03-01. See http://lat.inf.tu-dresden.de/research/reports.html. Germany: Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, 2003.

[Baa03b] Franz Baader. "Terminological Cycles in a Description Logic with Existential Restrictions." In: Jan. 2003, pp. 325–330.

[Baa99] Franz Baader. "Logic-Based Knowledge Representation". In: *Artificial Intelligence Today: Recent Trends and Developments*. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 13–41. ISBN: 3540664289.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. "Pushing the EL Envelope". In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. IJCAI'05. Edinburgh, Scotland: Morgan Kaufmann Publishers Inc., 2005, pp. 364–369.

[BHS04] Franz Baader, Ian Horrocks, and Ulrike Sattler. "Description Logics". In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 3–28. ISBN: 978-3-540-24750-0. DOI: 10.1007/978-3-540-24750-0_1. URL: https://doi.org/10.1007/978-3-540-24750-0_1.

[BK06] Franz Baader and Ralf Küsters. "Nonstandard Inferences in Description Logics: The Story So Far". In: July 2006, pp. 1–75. ISBN: 0-387-28688-8. DOI: 10.1007/0-387-31072-X_1.

[BKM98] F. Baader, R. Küsters, and R. Molitor. *Computing Least Common Subsumers in Description Logics with Existential Restrictions*. LTCS-Report LTCS-98-09. See http://www-lti.informatik.rwth-aachen.de/Forschung/Papers.html. Germany: LuFG Theoretical Computer Science, RWTH Aachen, 1998.

[ET12] Andreas Ecke and Anni-Yasmin Turhan. "Optimizations for the Role-Depth Bounded Least Common Subsumer in EL+". In: *OWLED*. 2012.

[Gra+08] Bernardo Cuenca Grau et al. "OWL 2: The next step for OWL". English. In: *Web Semantics* 6.4 (Nov. 2008), pp. 309–322. ISSN: 1873-7749. DOI: 10.1016/j.websem.2008.05.001.

[HB09] Matthew Horridge and Sean Bechhofer. "The OWL API: A Java API for Working with OWL 2 Ontologies". In: *Proceedings of the 6th International Conference on OWL: Experiences and Directions - Volume 529*. OWLED'09. Chantilly, VA: CEUR-WS.org, 2009, pp. 49–58.

[HHK95] M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. "Computing simulations on finite and infinite graphs". In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. 1995, pp. 453–462. DOI: 10.1109/SFCS.1995.492576.

[HM01] Volker Haarslev and Ralf Möller. "High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study". In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'01. Seattle, WA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 161–166. ISBN: 1558608125.

[Hof21] Thomas Hofweber. "Logic and Ontology". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2021. Metaphysics Research Lab, Stanford University, 2021.

[HSG15] R. Hoehndorf, P. N. Schofield, and G. V. Gkoutos. "The role of ontologies in biological and biomedical research: a functional perspective". English. In: *Briefings in Bioinformatics* 16.6 (Nov. 2015), pp. 1069–1080. ISSN: 1467-5463. DOI: 10.1093/bib/bbv011.

[KH17] Adila Krisnadhi and Pascal Hitzler. "Description Logics". In: *Encyclopedia of Social Network Analysis and Mining*. Ed. by Reda Alhajj and Jon Rokne. New York, NY: Springer New York, 2017, pp. 1–10. ISBN: 978-1-4614-7163-9. DOI: 10.1007/978-1-4614-7163-9_108-1. URL: https://doi.org/10.1007/978-1-4614-7163-9_108-1.

[KKS12] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. "ELK Reasoner: Architecture and Evaluation". In: *Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE-2012), Manchester, UK, July 1st, 2012*. Ed. by Ian Horrocks, Mikalai Yatskevich, and Ernesto Jiménez-Ruiz. Vol. 858. CEUR Workshop Proceedings. CEUR-WS.org, 2012. URL: http://ceur-ws.org/Vol-858/ore2012_paper10.pdf.

[KM02] Ralf Küsters and Ralf Molitor. "Approximating Most Specific Concepts in Description Logics with Existential Restrictions". In: *AI Commun.* 15.1 (Jan. 2002), pp. 47–59. ISSN: 0921-7126.

[Kri] Francesco Kriegel. *De-tu-dresden-inf-lat/Abox-repairs-WRT-static-tbox*. URL: https://github.com/de-tu-dresden-inf-lat/abox-repairs-wrt-static-tbox.

[LPP12] Manh Ha Le, Van Trung Pham, and Thi Ha Duong Phan. "A Polynomial-Time Algorithm for Reachability Problem of a Subclass of Petri Net and Chip Firing Games". In: *2012 IEEE RIVF International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future*. 2012, pp. 1–6. DOI: 10.1109/rivf.2012.6169852.

[LW10a] Carsten Lutz and Frank Wolter. "Deciding inseparability and conservative extensions in the description logic E L". In: *J. Symb. Comput.* 45 (Feb. 2010), pp. 194–228. DOI: 10.1016/j.jsc.2008.10.007.

[LW10b] Carsten Lutz and Frank Wolter. "Deciding inseparability and conservative extensions in the description logic EL". In: *Journal of Symbolic Computation* 45.2 (2010). Automated Deduction: Decidability, Complexity, Tractability, pp. 194–228. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2008.10.007. URL: https://www.sciencedirect.com/science/article/pii/S0747717109001126.

[Mat+14] Nicolas Matentzoglu et al. "The Manchester OWL Repository: System Description". In: ISWC-PD'14. Riva del Garda, Italy: CEUR-WS.org, 2014.

[Nad] Mohamed Ali Nadeem. *Maliabd-al-Majid/MSC: Polynomial Algorithm for finding the most specific concept*. URL: https://github.com/maliabd-al-majid/Msc.

[Noy+09] Natasha Noy et al. "BioPortal: Ontologies and Integrated Data Resources at the Click of a Mouse". In: *Nucleic acids research* 37 (June 2009), W170–3. DOI: 10.1093/nar/gkp440.

[Par+17] Bijan Parsia et al. "The OWL Reasoner Evaluation (ORE) 2015 Competition Report". In: *Journal of Automated Reasoning* 59 (Dec. 2017). DOI: 10.1007/s10817-017-9406-8.

[PT11] R. Peñaloza and A.-Y. Turhan. "A Practical Approach for Computing Generalization Inferences in $\mathcal{EL}$". In: *Proceedings of the 8th European Semantic Web Conference (ESWC'11)*. Ed. by Marko Grobelnik and Elena Simperl. Lecture Notes in Computer Science. Springer-Verlag, 2011.

[SS09] Steffen Staab and Rudi Studer. *Handbook on ontologies*. Berlin: Springer, 2009. ISBN: 9783540926733 3540926739. URL: http://public.eblib.com/choice/publicfullrecord.aspx?p=571805.

[Tur13] Anni-Yasmin Turhan. "Introductions to Description Logics – A Guided Tour". In: *Reasoning Web. Semantic Technologies for Intelligent Data Access: 9th International Summer School 2013, Mannheim, Germany, July 30 – August 2, 2013. Proceedings*. Ed. by Sebastian Rudolph et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 150–161. ISBN: 978-3-642-39784-4. DOI: 10.1007/978-3-642-39784-4_3. URL: https://doi.org/10.1007/978-3-642-39784-4_3.

[ZT13] Benjamin Zarrieß and Anni-Yasmin Turhan. *Most Specific Generalizations w.r.t. General $\mathcal{EL}$-TBoxes*. LTCS-Report 13-06. See http://lat.inf.tu-dresden.de/research/reports.html. Dresden, Germany: Chair of Automata Theory, Institute of Theoretical Computer Science, Technische Universität Dresden, 2013.