Quantitative Variants of Language Equations and their Applications to Description Logics

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Pavlos Marantidis, M.Sc.

geboren am 29. Dezember 1990 in Thessaloniki, Griechenland

verteidigt am 25. Juli 2019

Gutachter:
Prof. Dr.-Ing. Franz Baader
Technische Universität Dresden

Prof. Paliath Narendran
University at Albany – SUNY

Dresden, im September 2019

Abstract

Unification in description logics (DLs) has been introduced as a novel inference service that can be used to detect redundancies in ontologies, by finding different concepts that may potentially stand for the same intuitive notion. Together with the special case of matching, they were first investigated in detail for the DL $\mathcal{FL}_0$, where these problems can be reduced to solving certain language equations. In this thesis, we extend this service in two directions. In order to increase the recall of this method for finding redundancies, we introduce and investigate the notion of approximate unification, which basically finds pairs of concepts that "almost" unify, in order to account for potential small modelling errors. The meaning of "almost" is formalized using distance measures between concepts. We show that approximate unification in $\mathcal{FL}_0$ can be reduced to approximately solving language equations, and devise algorithms for solving the latter problem for particular distance measures. Furthermore, we make a first step towards integrating background knowledge, formulated in so-called TBoxes, by investigating the special case of matching in the presence of TBoxes of different forms. We acquire a tight complexity bound for the general case, while we prove that the problem becomes easier in a restricted setting. To achieve these bounds, we take advantage of an equivalence characterization of $\mathcal{FL}_0$ concepts that is based on formal languages. In addition, we incorporate TBoxes in computing concept distances. Even though our results on the approximate setting cannot deal with TBoxes yet, we prepare the framework that future research can build on. Before we journey to the technical details of the above investigations, we showcase our program in the simpler setting of the equational theory ACUI, where we are able to also combine the two extensions. In the course of studying the above problems, we make heavy use of automata theory, where we also derive novel results that could be of independent interest.

Acknowledgements

A great lot of people have played a significant role during my time as a PhD student. The present dissertation is the result of all these interactions. They are too many to enumerate, but I really want to distinguish some of them.

First and foremost, my supervisor Franz Baader. I am deeply grateful for everything he has done for me these past 4 years, from helping me with my research, to being patient and always relieving my stress when I had trouble getting results, and to finding me extra funding for as long as I needed to finish this thesis. One of the top researchers I have encountered, and a really interesting person to be around. I am really glad and proud to be counted as one of his (former) PhD students. Together with him, I would also like to thank all my coauthors for the opportunity to work together.

The research training group QuantLA. Even more important than the financing part, was the environment provided. I greatly benefitted from the interaction between different scientific communities, as well as the fact that whenever I needed an expert on some topic I could always find one around. A special bond is created when you put people to work alognside with the common goal of pursuing a PhD; the program felt very much like a family, and I am thankful for the opportunity they gave me. Furthermore, the final strecth of my PhD time was funded by HAEC.

George Rahonis, the single person responsible for me applying to come to Dresden, a mentor and a valuable friend.

The colleagues/friends from the Chair for Automata Theory, both current and old, who made the working environment a pleasant place: Oliver, Jakub, Willi, Filippo, Patrick, Christian, Anni, Julian, Marcel, Andreas, Ismail. Special thanks to Daniel Borchmann, who greatly helped me from the first day I arrived in Dresden with bureaucracy, research, and advice of every kind. He was a very important member of the Chair and his departure from the university left an unfillable gap. Furthermore, to our secretaries, Kerstin, Sandy, and Kati for always being there to deal with the intricate bureaucracy, and often going the extra mile to help me.

My best buddies in Dresden, Max and Antoine. We were brought together by necessity due to QuantLA, but we connected very fast and became inseparable. They have always been there to hear me complain and help me go on. I am not sure if I would be able to pull through without their support and friendship. I hope this relation will survive the test of time and distance.

All my friends that stood by me (from a greater distance) during this time: My buddies from Thessaloniki, Giannis, Christos, Iro, and especially Souvas, whose contribution during the final stretch of writing this thesis was invaluable. Fellow doctoral students around the globe, Vangelis, Madeline, Vassiliki. The numerous great friends I have made in Berlin. Instead of naming them individually, I will only refer to most prominent among them, the one who brought all of us together, my spiritual advisor, p.Emmanuel. His guidance and friendship always helped me to push through.

Finally, my oversized family: my parents Fotis and Athanasia, and my siblings Maria (and her family), Grammatiki (and her family), Angeliki, Christina, Danae, Georgia, Loida, Stavros and Titos. Their support all this time has been invaluable. They are my compass, and I cannot express my love and gratitude to them.

Contents

Chapter 1

Introduction

Description Logics (DLs) [BCM+03] are a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-undestood way. They allow to define the relevant concepts of the domain by introducing *concept names* and then use these names to specify properties of objects occuring in the domain by providing *concept descriptions*, forming a knowledge base. As the name indicates, one of the characteristics of DLs is that they are equipped with a formal, logic-based semantics. Therefore, they allow for *reasoning*, i.e., inferring implicitly represented knowledge from the information that is explicitly contained in the knowledge base. In particular, one can, for example, check for subconcept-superconcept relation or equivalence between two given concept descriptions.

If a knowledge base is maintained by different knowledge engineers, one needs other inference services in order to support and maintain the knowledge base. In particular, it is useful to be able to detect multiple definitions of the same intuitive concept. Since different knowledge engineers might use different names for the "same" primitive concept, the standard equivalence test may not be adequate to check whether different descriptions refer to the same notion. Unification in DLs tackles this problem by allowing concept names to be replaced by appropriate concept descriptions before testing for equivalence. If such a replacement that makes the two concept descriptions equivalent exists, it is called a *unifier*, and potentially provides a definition for the concept name it replaced. Matching is a special case of unification, where one of the concept descriptions is considered to be fixed, i.e., the concept names appearing in it cannot be replaced by other concept descriptions.

However, this procedure might not work if there are small modelling errors, or even slight discrepancies in the concept descriptions under examination. In particular, it might not be possible to make the concept descriptions equivalent, but there might still be a replacement that makes them quite similar. Such an *approximate* approach can increase the recall of the service.[1] Furthermore, whereas two concepts might not be unifiable, the presence of background terminological knowledge might reveal connections between the constituents of the concepts that will allow for a replacement that renders the concepts equivalent. Even though background terminological knowledge plays an important role in standard inference services, its integration in unification is rather limited.

The aim of this thesis is to extend unification in the DL $\mathcal{FL}_0$ in the two aforementioned ways. On the one hand, we introduce and investigate the notion of approximate unification. Basically, to formalize approximate unification, we first need to fix the notion of a distance

[1] Recall is a measure of performance of an information retrieval system. Often called *sensitivity*, it is the fraction of relevant instances that have been retrieved over the total amount of relevant instances [PKB55].

between concept descriptions. An approximate unifier is then supposed to make this distance as small as possible, instead of trying to make the concepts equivalent.

On the other hand, we try to incorporate TBoxes, i.e., the background knowledge, in this inference service. Decidability of unification in $\mathcal{FL}_0$ in the presence of TBoxes has been open for almost two decades now, and it still seems to be a quite difficult problem. Therefore, our goal will be less ambitious: we will prove decidability and derive complexity bounds for matching with TBoxes.

In order to obtain some initial results and gain some intuition, we will start with investigating these problems in a simpler setting. There, we are not only able to study these extensions in depth, but also to combine them.

There is a well-documented relation between $\mathcal{FL}_0$ concept descriptions and formal languages, a link that has been profoundly utilized in the past [Baa96; BN01; Pen15; BFP18]. In particular, Baader and Narendran [BN01] proved that $\mathcal{FL}_0$ unification reduces to solvability of certain language equations, using tree automata to tackle the latter problem. Our approach follows the same pattern, considering variations of these language equations, and our techniques involve employing and developing notions from automata theory. Even though we obtain technical results to tackle the problem at hand, these contributions are of independent interest in their area.

1.1 Description Logics

Knowledge Representation (KR) [HLP08] is the field of artificial intelligence dedicated to representing information about the world in a form that a machine can utilize to solve complex tasks. In particular, it often incorporates findings from logic to automate various kinds of reasoning. Description Logics is a family of KR logic-based formalisms that are descended from so-called "structured inheritance networks" [Bra77; Bra78], and were introduced to overcome the ambiguities of the previous approaches of semantic networks and frames.

The basic vocabulary of DLs consists of concepts, which denote sets of individuals, and roles, which denote binary relationships between individuals. In addition to these, all DLs allow their users to construct complex descriptions of concepts and roles. The language for building such descriptions is characteristic for each DL. For example, given the concept names Woman, Rich, Smart and the role name child, the expression Woman$\sqcap\forall$child.(Rich$\sqcap$Smart) denotes the concept of a woman all the children of whom are both rich and smart.

A DL system offers services to reason about concept descriptions. A typical reasoning task is to determine whether a description is more general than another one, that is, whether the first subsumes the second. In our previous example, every individual that belongs to the stated concept is also an instance of the concept Woman$\sqcap\forall$child.Rich, and even more generally, of the concept Woman.

DLs provide their users with ways of stating terminological axioms in so-called TBoxes. The simplest kind of TBoxes are called acyclic TBoxes, which consist of concept definitions without cyclic dependencies among the defined concepts. Basically, such a TBox introduces abbreviations for complex concept descriptions. When the dependency restriction is lifted, we obtain cyclic TBoxes. General TBoxes use so-called general concept inclusions (GCIs) to state subconcept-superconcept constraints between concepts. One could for example express that every Woman is also a Human by introducing the GCI Woman$\sqsubseteq$Human. Reasoning

services become even more meaningful when paired with a TBox. In our running example, using the above GCI we could further infer that every instance of the original concept is also a Human.

Investigating the computational complexity of reasoning for different sets of constructors has been a guiding theme in DL research. In the paper "The tractability of subsumption in Frame-Based Description Languages" [BL84], which is regarded as the origin of research on Description Logics, Brachman and Levesque argued that there is a tradeoff between the expressiveness of the representation language and the difficulty of reasoning over the representations built using that language. In other words, the more expressive the language, the more expensive reasoning becomes. They also provided a first example of this tradeoff by analyzing the language $\mathcal{FL}$ (Frame Language), which included conjunction of concepts, value restrictions, a simple form of existential restriction, and a construct called role restriction. They showed that for this language subsumption is a coNP-hard problem, while removing role restrictions, obtaining a language they called $\mathcal{FL}^-$, the problem becomes tractable.

Later, Nebel in [Neb90] investigated an even smaller fragment of $\mathcal{FL}^-$, which he called $\mathcal{TL}$ and contained only conjunction and value restrictions. He proved that the addition of background terminological knowledge, in particular what he calls *acyclic terminologies*, which nowadays comes by the name *acyclic TBoxes*, causes the complexity of subsumption to increase to coNP-complete. In fact, he describes this language as "a minimal terminological representation language that is a subset of every useful terminological language". The same fragment was later used by Baader [Baa96] to show that *cyclic terminologies* (cyclic TBoxes) cause the complexity of subsumption reasoning to increase even more, as the problem becomes PSPACE-complete. He did not reuse the same name, but rather called this description language $\mathcal{FL}_0$, since it is the least possible fragment of $\mathcal{FL}$ that still makes sense to investigate. This has been the name that stuck and is being used until the current date. In the passing of time, though, the language has come to contain the *top concept* as well. Even though this constructor does not significantly increase the expressivity of the DL, this addition most probably happened in order to ensure that the corresponding equational theory is monoidal [BN96], and hence be able to make use of the relevant results. Furthermore, this way $\mathcal{FL}_0$ better matches the dual basic DL $\mathcal{EL}$, which allows for conjunction, *existential restrictions*, and the top concept. In this thesis, we will predominantly deal with this (current) version of $\mathcal{FL}_0$.

As DLs became increasingly used, researchers investigated a multitude of additional reasoning tasks that are intended to make DLs more usable in various applications. This resulted in the development of several new procedures, including, among many others, computing least common subsumers and concept difference, and also the engagement of techniques from other areas, like conservative extensions, forgetting, unification and matching. These last two are the main focus of this thesis.

1.2 Unification

Unification theory [BS94] is a field of its own interest, predating Description Logics. Unification is a fundamental process upon which many methods for automated deduction are based. Very generally speaking, it deals with solving equations. Given two symbolic expressions, unification tries to identify them by replacing certain sub-expressions by other expressions.

To be more concrete, consider the terms $s = f(a, x)$ and $t = f(y, b)$, where f is a binary function symbol, a, b are constant symbols, and x, y are variable symbols. The unification problem for s and t asks whether it is possible to replace the variables x, y occurring in s and t by other terms such that the two terms obtained this way are equal. In this example, if we substitute x with b and y with a, we say that the two terms are *unified*, since they both become $f(a, b)$. The substitution that made this happen is called a *unifier*.

Note that different occurences of the same variable in a unification problem must always be replaced by the same term. For this reason, s cannot be unified with $t' = f(x, b)$, since this would require the occurence of x in s to be substituted with b, while the occurence of x in t' should be substituted with a.

Instead of requiring that the terms are made syntactically equal, one can try to make them equivalent modulo a given equational theory E. This type of unification is called E-unification or equational unification. Consider, for example, the theory $E = \{f(a, c) \approx f(c, b)\}$, where c is also a constant, and the terms s and t' we examined above, which we noted that are not unifiable. By replacing x by c, we obtain the terms $f(a, c)$ and $f(c, b)$, which are not the same, but they are *equivalent* modulo E, and the substitution is an *E-unifier*.

In fact, one of the major topics in unification theory is to investigate the properties of equational theories in a systematic way [BS01; BS94; Sie89]. More concretely, one can consider the decision problem, i.e., the question whether a given E-unification problem has an E-unifier or not. For decidable problems one can then research their complexity. Instead of just deciding unifiability, one often also wants to compute E-unifiers in case they exist. In fact, it is interesting for some applications, like automated deduction and term rewriting, to compute a minimal complete set of E-unifiers, in the sense that every unifier of the problem is an instance of a unifier in the set (completeness), and two different unifiers in the set are incomparable (minimality). Of particular interest is the *unification type* of an equational theory, which counts how large such sets can get. However, we will not deal with these notions in the currect thesis.

As we will discuss later, unification in Description Logics can be viewed as a unification problem modulo an appropriate equational theory. The notion of unification was introduced in description logics as a tool for recognizing redundant concept descriptions. Basically, the idea is the following. Assume that we have two concept descriptions C and D that we suspect to be different formalizations of the same intuitive concept. One might think that it is enough to test whether C and D are equivalent, i.e., describe the same set of objects in every interpretation. This is sometimes indeed sufficient, since, for example our running concept Woman⊓∀child.(Rich⊓Smart) is equivalent to the concept Woman⊓∀child.Rich⊓∀child.Smart, even though the corresponding terms are not (syntactically) the same. This approach, however, disregards the possibility that the two descriptions may employ different names for primitive concepts or be modeled on different levels of granularity, where a concept name in one description actually corresponds to a complex subdescription in the other one. Take for example the concepts Woman⊓∀child.(Rich⊓Smart) and Human⊓Female⊓∀child.Prosperous, where Propsperous is a concept name; even though the human eye would probably recognize these as describing the same notion, a test for equivalence would return back negative. This problem might be overcome by testing the descriptions for *unifiability* modulo equivalence rather than equivalence itself. In the previous example, if we regard Prosperous as a variable, and replace it by Rich⊓Smart, we obtain two equivalent concept descriptions. In this application, the existence of a finite

minimal complete set of unifiers does not appear to be relevant. Instead, one is interested in computing *ground* unifiers, i.e., unifiers that replace all variables by concept descriptions not containing variables. If σ is a ground unifier of the concept descriptions C and D, we essentially obtain that, if we were to add concept definitions $X \equiv \sigma(X)$ for all the concept names X that were viewed as variables in the unification problem, then we would make the two descriptions equivalent w.r.t. these definitions. Of course, the knowledge engineer needs to check whether these definitions really make sense within the application domain that is modeled with these concepts.

In this setting, unification was first introduced and investigated for the description logic $\mathcal{FL}_0$ by Baader and Narendran [BN01]. In fact, they showed that unification in $\mathcal{FL}_0$ corresponds to unification modulo the equational theory ACUIh, i.e., that of a binary associative, commutative, and idempotent function symbol with a unit and several unary function symbols that behave like homomorphisms[2]. Later, Baader and Morawska [BM10] investigated unification in the dual logic $\mathcal{EL}$, to which corresponds the equational theory ACUIm, which differs from ACUIh in that instead of homomorphisms, the unary function symbols are monotone operators[3].

The common subtheory of the two, ACUI, that is, the above theories without the unary function symbols, is one of the first equational theories for which the complexity of testing solvability of unification problems was investigated in detail [KN92]. In particular, terms modulo ACUI correspond to subsets of a base set of elements,[4] which makes reasoning and analysis in this setting rather simple and intuitive. Before we extend unification in $\mathcal{FL}_0$ by investigating approximate unification and the addition of TBoxes, we believe that ACUI is a good testing ground for obtaining and demonstrating some initial intuitive results. In fact, the corresponding description logic has appeared in the DL literature under the name $\mathcal{L}_0$ in [LT12] and $\mathcal{A}$ in [BWH05][5] with the same purpose: to preliminarily test some ideas before extending them to more expressive languages.

1.3 Approximate services

DLs allow the same thing to be described in different ways. Two concepts can be syntactically different, yet semantically equivalent as long as they satisfy the same properties. Recall for example the concept descriptions we provided earlier. However, since the semantics of traditional DLs is based on classical first-order logic, this interpretation of equivalence is rather strict. Either two concepts satisfy the *exact* same set of properties and they are equivalent, or not; there is no in-between. Nevertheless, we might often be interested in not only whether two concepts are equivalent, but also if they are "close enough".

[2] In [BM10] this is refered to as the equational theory of idempotent Abelian monoids with several homomorphisms, which is essentially a different name for the same notion.

[3] The name provided in [BM10] is the equational theory of semilattices with monotone operators $SLmO$, which again is the same equational theory.

[4] Recall that, given their connection with $\mathcal{FL}_0$ concept descriptions, terms modulo ACUIh correspond to formal languages, i.e., more structured objects, and are hence of a more complex nature.

[5] To be exact, this DL only allows for conjunction of concept names, while the language corresponding to ACUI would also allow for the top concept. The inclusion of the latter, however, would not have any serious repercussions.

The term *approximation* first appeared in Description Logics in a paper by Baader, Küsters, and Molitor [BKM00], and investigated by Brandt, Küesters, and Turhan [BKT02], where it described the inference problem of rewriting a concept description C from one DL into a concept description D from another, usually not equally expressive DL. Most often, it is not possible to obtain a concept description that is *equivalent* to the original one. In this case, the goal is to obtain a concept description that is semantically "as close as possible" to the original one. To achieve this, the resulting concept D should either be a minimal (w.r.t. subsumption) concept that subsumes C, or a maximal concept that is subsumed by C. Intuitively, this is a primitive way to obtain a concept that has the least distance from C. However, if there is no such minimal (resp. maximal in the latter case) concept, it might suffice to obtain a concept that is "not too far away", i.e., closer than a given threshold. To formally define this notion, one should first employ a mechanism capable of assessing how far apart two concepts are.

The idea of measuring *distance between concepts* has received considerable attention in several domains, including psychology, cognitive science, and computational linguistics. In the context of DLs, research into such measures has been started in [BWH05], where the authors originally translate approaches in other formalisms onto the inexpressive logic $\mathcal{A}$[6] before they discuss ways to extend these ideas to more expressive logics.

Since then, there have been several investigations on how to define measures of distance between concepts in DLs [Jan06; LT12; Eck17; RS15; Sun13; Bra04], which paved the way for the exploration of *approximate* versions of existing reasoning problems, like relaxed instance queries [EPT15], prototypical definitions [BE16], and threshold concepts. [BBF15] In this spirit, we investigate approximate unification and matching.

Note that the term approximation is often used to describe approaches that try to speed up reasoning by employing inference techniques that might not be complete [PRZ16]. Our use of the term, however, describes the attempt to extend the range of admissible answers to queries or admissible elements of concept descriptions or admissible solutions to a unification problem, and it is inherently connected to the notion of distance between concepts.

Before we start investigating approximate unification we will talk about such measures in more depth. A major concern in some of the aforementioned papers is how the available background knowledge (TBoxes) can be taken into consideration. Our approach will demonstrate how this is possible for the DL $\mathcal{FL}_0$.

1.4 From concepts to languages and automata

Concepts are essentially sets of attributes that individuals should satisfy. For $\mathcal{FL}_0$, due to the particular semantics, these attributes admit a more structured representation: every attribute can be viewed as a word over the alphabet of role names. As demonstrated throughout the literature, every $\mathcal{FL}_0$ concept description can be reduced to a tuple of formal languages over the alphabet of role names. This reduction is equivalence preserving, in the sense that equivalent descriptions correspond to the same tuple and vice versa. One can find traces of this idea already in the subsumption algorithm for $\mathcal{FL}^-$ in [BL87], and this correlation has been a guiding theme in research for $\mathcal{FL}_0$ (to name but a few, consider the publications

[6] Recall that this is the DL corresponding to the equational theory ACUI.

[Neb90; Baa96; BFP18]). This thesis is no exception to the aforementioned rule. Virtually every result about $\mathcal{FL}_0$ is derived by taking advantage of the reduction to formal languages.

Utilizing this connection, Baader and Narendran in [BN01] were able to reduce unification in $\mathcal{FL}_0$ to solving certain language equations. In particular, these are equations where both the constants occurring in the equations and the solutions are finite formal languages, and the only allowed operations are union and one-sided concatenation. In turn, solving these equations was reduced to testing certain tree automata for emptiness. We will extend this approach to deal with approximate unification. In fact, by linking distance functions on concept descriptions with distance functions on languages, we can reduce approximate unification in $\mathcal{FL}_0$ to approximately solving language equations. In order to reduce this problem to a problem for tree automata, we do not employ the original construction of [BN01], but the more sophisticated one from the more language-equation-centered [BO13].

The presence of general TBoxes may lead the set of attributes a concept satisfies to become infinite. Hence TBoxes can result into the languages corresponding to an $\mathcal{FL}_0$ concept description to be infinite. Nevertheless, these languages can be used to characterize equivalence in this setting as well. In fact, recently published [Pen15] and [BFP18] provide an effective representation of these tuples, and these results will in fact play an important role when we consider TBoxes in our investigation.

1.5 Structure of the Thesis

In the following we will give a short outline for the remainder of this thesis. At the end of this section, we also provide a list of the publications that most of the results in this thesis are based on.

Chapter 2 serves as a warm-up and a demonstration of our program for extending unification. In particular, it focuses on unification modulo the equational theory of an associative, commutative, and idempotent binary function symbol with unit ACUI, which is a subtheory of (the equational theory corresponding to) $\mathcal{FL}_0$. Initially, we formally introduce unification in this simple setting and describe the approach of [KN92] that proved the problem to be decidable in polynomial time. Next, we consider approximate unification in ACUI w.r.t. three different measures. We prove that the complexity increases from P to NP-complete when going from exact ACUI-unification to the approximate case for two of the measures, whereas it stays in P for the third one. We proceed to investigate how the complexity of unification is affected by adding finite sets of ground identities G. We are able to show that the problem remains solvable in P, while we also derive NP-completeness for the problem of general unification. In the last part of this chapter, we combine the two extensions by considering approximate unification modulo ACUIG, i.e., ACUI extended with a finite set of ground identities G. For two of the measures, the NP-completeness results transfer from ACUI to ACUIG. For the third measure, the problem can either remain in P or become NP-complete, depending on the ground theory G. The results of this chapter are based and extend [BMO16b] and [BMM18]. The first paper introduces approximate ACUI-unification, while the second investigates unification modulo ACUIG.

Chapter 3 deals with technical details from formal language and automata theory. It presents the basic relevant notions, starting from words, languages and language operations, and describing ways to specify languages, in particular deterministic and nondeterministic

finite automata and regular expressions, and how we can transit from one model to the other. We then study language distances reviewing existing results. We distinguish three distances in particular, d_0, d_1 and d_2, that will be of special interest in the rest of the thesis. Next, we transit from words to trees. We introduce infinite trees and describe how we can utilize them to describe tuples of (word) languages. Looking for a finite representation of such trees, we will restrict to regular trees, and present tree automata models, in particular Büchi and looping tree automata that can be used to specify trees. After we extensively study how LTAs can represent infinite trees and we present how we can transform a tuple of word automata into a tree automaton, we describe how trees can be used to define languages. Afterwards, we introduce weighted looping tree automata (wLTAs) with discounting that assign weights from a semiring to trees, and we study how they can be used (together with the representation of tuples of languages as trees) to express language distances. Finally, we investigate computation of the behavior of a weighted tree automaton, that is, given an input tree, what value (weight) the automaton assigns to the tree. We prove that for regular trees and wLTA over the semiring $\mathbb{R}_{\text{inf}}$ with nondecreasing or contracting discounting, it is possible to compute this value in time polynomial in the size of the input. The novel results in this chapter, in particular representation of languages by trees and computability of wLTAs are based on [BFM17].

Chapter 4 is the first chapter that actually deals with Description Logics. We first present the syntax and semantics of the Description Logic $\mathcal{FL}_0$ and we describe TBoxes and the problem of subsumption between concepts. We next present how subsumption is characterized by inclusion between languages that are finite for the empty TBox, but may become infinite otherwise. Afterwards we give the formal definition of unification in $\mathcal{FL}_0$ and describe the approach from [BN01] that reduces this problem to solving language equations of a certain kind. Finally, we describe how these equations can be solved in ExpTime, by describing the approach from [BO13].

Chapter 5 focuses on distance measures between $\mathcal{FL}_0$ concepts (CDMs) in the presence of a TBox. In particular, utilizing the connection between $\mathcal{FL}_0$ concepts and (tuples of) formal languages we provide a modular framework that shows how a language distance can be properly combined with a function to define a CDM. After we instantiate this framework with the language distances d_0, d_1 and d_2, we review what further properties that have been investigated in the literature our framework satisfies. We conclude the chapter by investigating computability of CDMs obtained in this way. The modular setup of the framework allows for a transparent composition of the ingredients, which are a way to derive from a given concept the corresponding tuple of languages, and how to compute the given language distance. We prove that, under certain assumptions, w.r.t. a CDM obtained by our framework, given a TBox, the distance between two $\mathcal{FL}_0$ concepts can be computed in exponential time. The results of this chapter are based on [BFM17].

Chapter 6 investigates approximate unification. After we formally define the problem using CDMs, we prove that the decision version of the problem can be reduced to the computational version of it. Next, we define the problem of approximately solving language equations based on language distances. We proceed to prove that for CDMs that are defined by the framework we introduced in Chapter 5, approximate $\mathcal{FL}_0$ unification can be reduced to approximately solving language equations, replicating the approach for exact unification. In order to approximately solve language equations, we consider language distances that can be expressed by a wLTA, and provide a reduction to problems on tree automata. Overall, we

obtain that approximate unfication w.r.t. such distances can be decided in ExpTime. Finally, we focus on the cases of d_1 and d_2 to provide a matching lower bound. The results of this chapter are mainly based on [BMO16a].

Chapter 7 restricts to the special case of approximate matching in $\mathcal{FL}_0$. Initially, we review the approach of [BN01] on how the classical version can be solved in polynomial time since it reduces to language equations that are easier to solve. We then prove that for a wide variety of distances the problem is in NP. We proceed to compute exact complexity bounds for certain distances. In particular, for d_1 we prove that the problem is still solvable in polynomial time, while for d_0 and d_2 it is NP-complete. To obtain the latter hardness result we introduce a version of a satisfiability problem, that we call Max-±Pos-SAT, which we also prove to be an NP-complete problem. Some of the results from this chapter have appeared in [BM17].

Chapter 8 deals with matching in $\mathcal{FL}_0$ in the presence of general TBoxes. The main result is ExpTime-completeness of the problem. Since ExpTime-hardness of this problem is clear, the first main contribution of this chapter is thus to show the ExpTime upper bound. We do this by first showing an ExpTime upper bound for the problem of testing whether an $\mathcal{FL}_0$ matching problem has a matcher in the extended logic $\mathcal{FL}_{\text{reg}}$. Our proof of the ExpTime upper bound depends on a fine-grained analysis of the complexity of subsumption of $\mathcal{FL}_{\text{reg}}$ concept descriptions w.r.t. an $\mathcal{FL}_0$ TBox. The second step is then to show that an $\mathcal{FL}_0$ matching problem has an $\mathcal{FL}_0$ matcher iff it has an $\mathcal{FL}_{\text{reg}}$ matcher. The second main contribution of this chapter is to show that the complexity of the matching problem can be lowered from ExpTime to PSpace if one considers TBoxes of a restricted form where the role depth on the left-hand side of a GCI is not larger than the role depth on the right-hand side. The results of this chapter appeared in [BFM18].

Chapter 9 summarizes the contributions of this thesis and discusses directions for future work.

Throughout this thesis we deal with several notions from metric topology and calculus. After the bibliographic references we provide an appendix with the relevant basic definitions and results.

[BFM17] Franz Baader, Oliver Fernández Gil, and Pavlos Marantidis: 'Approximation in Description Logics: How Weighted Tree Automata Can Help to Define the Required Concept Comparison Measures in $\mathcal{FL}_0$'. In *Proceedings of the 11th International Conference on Language and Automata Theory and Applications (LATA 2017)*, Umeå, Sweden. Edited by Frank Drewes, Carlos Martín-Vide, and Bianca Truthe. Volume 10168. Lecture Notes in Computer Science. Springer, 2017, pages 3–26 (cited on page 8).

[BFM18] Franz Baader, Oliver Fernández Gil, and Pavlos Marantidis: 'Matching in the Description Logic $\mathcal{FL}_0$ with respect to General TBoxes'. In *Proc. of the 22nd Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'18)*. Edited by Gilles Barthe, Geoff Sutcliffe, and Margus Veanes. Volume 57. EPiC Series in Computing. EasyChair, 2018, pages 76–94 (cited on page 9).

[BM17] Franz Baader and Pavlos Marantidis: 'Language equations for approximate matching in the Description Logic $\mathcal{FL}_0$'. In *Proceedings of the 31st International Workshop on Unification (UNIF'17)*. Edited by Adrià Gascón and Christopher Lynch. Oxford, UK, 2017 (cited on page 9).

[BMM18] Franz Baader, Pavlos Marantidis, and Antoine Mottet: 'ACUI Unification modulo Ground Theories'. In *Proceedings of the 32th International Workshop on Unification (UNIF 2018)*. Edited by Mauricio Ayala-Rincón and Philippe Balbiani. Oxford, UK, 2018, pages 37–41 (cited on page 7).

[BMO16a] Franz Baader, Pavlos Marantidis, and Alexander Okhotin: 'Approximate Unification in the Description Logic $\mathcal{FL}_0$'. In *Proc. of the 15th Eur. Conf. on Logics in Artificial Intelligence (JELIA 2016)*. Edited by Loizos Michael and Antonis C. Kakas. Volume 10021. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2016, pages 49–63 (cited on page 9).

[BMO16b] Franz Baader, Pavlos Marantidis, and Alexander Okhotin: 'Approximately Solving Set Equations'. In *Proceedings of the 30th International Workshop on Unification (UNIF'16)*. Edited by Silvio Ghilardi and Manfred Schmidt-Schauß. Porto, Portugal, 2016, pages 37–41 (cited on page 7).

Chapter 2

Extending unification modulo ACUI

In this warm-up chapter we demonstrate our program by extending unification in the equational theory ACUI by first investigating approximate unification, then by considering identities between ground terms, and finally by combining the two approaches.

2.1 Introduction

An important topic in unification theory [Sie89; BS94; BS01] is investigating the complexity of deciding solvability of unification problems Γ w.r.t. an equational theory E, i.e., for a fixed equational theory E one considers as input a set of equations between terms of the form $\Gamma = \{s_1 =^? t_1, \ldots, s_k =^? t_k\}$, and asks whether there exists a substitution σ such that $\sigma(s_i) =_E \sigma(t_i)$ holds for $i = 1, \ldots, k$. The complexity is then measured in the size of the unification problem Γ. The theory ACUI of an associative, commutative, and idempotent binary function symbol $+$ with unit $\mathbf{0}$ was one of the first equational theories for which the complexity of testing solvability of unification problems was investigated in detail [KN92]. Interestingly, it turned out that this complexity depends on which symbols are allowed to occur in Γ. In elementary E-unification, the terms in Γ may only contain variables and the constant and function symbols occurring in E, i.e., for $E =$ ACUI these terms are built using variables and $+, \mathbf{0}$. In E-unification with constants, additional free constants (i.e., constants not occurring in E) may be used, whereas in general E-unification both free constants and free function symbols may occur in Γ. Kapur and Narendran [KN92] have shown that elementary ACUI-unification and ACUI-unification with constants are polynomial, whereas general ACUI-unification is NP-complete.

As discussed in the Introduction of this thesis, our interest in ACUI-unification stems from the fact that the theory ACUI is a common subtheory of the equational theories corresponding to the Description Logics $\mathcal{FL}_0$ [Baa96] and $\mathcal{EL}$ [BKM99]: for $\mathcal{FL}_0$, ACUI is extended with unary function symbols that behave like homomorphisms and for $\mathcal{EL}$ the additional unary function symbols behave like monotone operators. Unification with constants in (the equational theory corresponding to) $\mathcal{FL}_0$ is known to be ExpTime-complete [BN01] and NP-complete in (the equational theory corresponding to) $\mathcal{EL}$ [BM09]. We believe that ACUI is thus a good testing ground before trying to extend these results to more general settings. Here, we consider extensions of ACUI-unification in two orthogonal directions, which we then bring together: generalizing unification to approximate unification and adding a finite ground theory to ACUI.

In approximate E-unification, one does not require that the left- and right-hand sides of the equations become equal modulo E, but only almost equal. The formal meaning of when a substitution approximately (i.e., "almost") solves a unification problem is formalized

using distance measures between terms that are tailored towards the equational theory E in question. In the present chapter, we consider approximate unification with constants in ACUI w.r.t. three different measures: the first considers the number of equations that are violated, the second the number of constants violating at least one equation, and the third counts the overall number of violations (see Section 2.3 for exact definitions of these measures). It turns out that in the first and the third case, the complexity increases from P to NP-complete when going from exact ACUI-unification with constants to the approximate case, whereas it stays in P for the second case. The main reason why we only consider unification with constants in this approximate setting is to align with the motivation from DL. Since concept names correspond to (free) constants, unification problems without the latter are of little interest. Furthemore, free function symbols would have no interpretation in the DL context, not to mention that there is no intuitive notion of how to define distance measures between terms that contain such symbols.

The results for unification in the Description Logics $\mathcal{FL}_0$ and $\mathcal{EL}$ respectively shown in [BN01] and [BM09] are restricted to the case where equivalence of concept descriptions (corresponding to equality modulo the respective equational theories mentioned above) is considered without a background knowledge base. It is not known how to extend these decidability and complexity results to unification in the presence of so-called general TBoxes, though for $\mathcal{EL}$ there are some positive results for unification with respect to a restricted form of TBoxes [BBM12], and also for matching with general TBoxes [BM14].[1] Since, from an equational theory point of view, general TBoxes correspond to finite sets of ground identities, we are interested in how the complexity of unification is affected by adding finite sets of ground identities. This will be the main focus of Section 2.4.

For the word problem, Marché proved in [Mar96] that ACUI remains decidable if it is extended with a finite set of ground identities, but no complexity bounds are given. This result actually holds for a signature possibly containing several ACUI symbols and free function symbols. In Section 2.4 we consider a restricted setting where the ground theory G is built using only one ACUI symbol and free constants.[2] In this case we can show that both the word problem and unification with constants remain solvable in P. Actually, the result for unification holds for unification problems with so-called constant restrictions, which allows us to employ the combination results from [BS96] to show that general unification in $\mathsf{ACUI} \cup G$ is NP-complete for any finite set of ground identities G satisfying the restrictions mentioned above. As opposed to the classical case, general unification in this setting is of particular interest for description logics in the following sense. Extending unification modulo $\mathsf{ACUI} \cup G$ by adding free function symbols is a first step towards unification modulo $\mathsf{ACUIh} \cup G$ ($\mathsf{ACUIm} \cup G$), i.e., unification in $\mathcal{FL}_0$ ($\mathcal{EL}$) in the presence of TBoxes, which adds unary function symbols that satisfy certain identities. Recall that unification in $\mathcal{FL}_0$ and $\mathcal{EL}$ in the presence of TBoxes is a long-standing open problem. The complexity upper bounds shown in Section 2.4 (in P for unification with constant restrictions and in NP for general unification) are actually somewhat stronger: these upper bounds hold even if we view G to

[1] In Chapter 8 we will show an analogous result by investigating matching in $\mathcal{FL}_0$ in the presence of (general) TBoxes.

[2] This is in line with the motivation from description logics: in $\mathcal{FL}_0$, $\mathcal{EL}$, and $\mathcal{L}_0$ we have conjunction as the sole binary operator, hence one ACUI symbol is enough.

be part of the input, i.e., measure the complexity in the size of Γ and G. NP-hardness already holds for any fixed finite set of ground identities G.[3]

In the last part of this chapter, we are combining the two extensions, by considering approximate unification modulo ACUIG, i.e., ACUI extended with a finite set of ground identities G. For the cases where one considers the number of violated equations or the overall number of violations, the NP-completeness results transfer from ACUI to ACUIG. For the remaining case where one considers the number of constants violating at least one equation, things become more interesting: we will show that there is a finite set of ground identities G such that approximate ACUIG-unification becomes NP-complete, but we will also exhibit a class of ground identities G for which this problem remains in P.

For the purposes of this chapter, we will only provide some basic definitions regarding the equational theory ACUI and ACUI-unification. The interested reader can find more details on equational theories and unification modulo equational theories in the relevant literature, e.g., in [BN98; BS01].

2.2 Unification modulo ACUI

In this section, we introduce the equational theory ACUI, characterize the word problem in ACUI using sets of constants, and recall the polynomial-time decision procedure for ACUI-unification by Kapur and Narendran [KN92], which is based on a translation of ACUI-unification problems into propositional Horn formulae. This translation will then be extended in the next section to deal with approximate unification.

Let $\Sigma = \{+, \mathbf{0}\}$ be the signature consisting of a binary function symbol $+$ and a constant symbol $\mathbf{0}$. We denote the equational theory that states that $+$ is an associative, commutative, and idempotent symbol with unit $\mathbf{0}$ by ACUI:

$$\mathsf{ACUI} := \{(x+y)+z \approx x+(y+z), x+y \approx y+x, x+\mathbf{0} \approx x, x+x \approx x\}.$$

Furthermore, let F be a countably infinite set of constants and V a countably infinite set of variables, where we assume that F, V, and Σ are pairwise disjoint. We call the elements of F *free constants* since they are not constrained by the identities of ACUI. We denote the set of terms built from Σ, F and V as $T_\Sigma(F,V)$, and the subset of ground terms, i.e., terms in $T_\Sigma(F,V)$ that do not contain variables, as $T_\Sigma(F)$. For example, if $a, b \in F$ and $x, y \in V$, then $x+y$ and $a+x$ belong to $T_\Sigma(F,V)$, and $a+b+b$ and $b+a+a$ are elements of $T_\Sigma(F)$. The latter two terms are actually equivalent modulo ACUI. More generally, two ground terms are equivalent modulo ACUI iff they contain the same free constants. To be more precise, given a ground term $t \in T_\Sigma(F)$, we denote the set of free constants occurring in t with $S(t)$. For example, $S(a+b+b) = \{a,b\} = S(b+a+a)$, and $S(a+\mathbf{0}) = \{a\} = S(a+a)$. The following result is well-known and also easy to see.

Lemma 2.1. *Let $s, t \in T_\Sigma(F)$. Then $s \approx_{\mathsf{ACUI}} t$ iff $S(s) = S(t)$.*

Before we can define ACUI-unification, we need to introduce the notion of a substitution. A *substitution* is a mapping $\sigma : V \to T_\Sigma(F,V)$ that is the identity for all but finitely many

[3] Recall that G corresponds to the TBox in the DL setting; in DL terminology when the TBox is considered part of the input we are talking about *combined complexity*, while when it is considered fixed we refer to *data complexity*.

variables. It can be homomorphically extended to a mapping from $T_\Sigma(F,V)$ to $T_\Sigma(F,V)$ in the obvious way, that is, $\sigma(c) = c$ for every $c \in F \cup \{\mathbf{0}\}$, and $\sigma(t_1 + t_2) = \sigma(t_1) + \sigma(t_2)$ for every $t_1, t_2 \in T_\Sigma(F,V)$.

Definition 2.2 (ACUI-unification problem with constants).

Input: *A finite system $\Gamma = \{s_1 \approx^? t_1, \ldots, s_k \approx^? t_k\}$ of equations between terms of $T_\Sigma(F,V)$.*

Question: *Is there a substitution σ such that $\sigma(s_i) \approx_{\mathsf{ACUI}} \sigma(t_i)$ for every $i = 1, \ldots, k$?*

Such a substitution is called an ACUI*-unifier of* Γ. ♢

The ACUI-unification problems introduced in the above definition are called unification problems *with constants* since they may contain additional free constants (i.e., the elements of F, which do not belong to Σ), but no additional free function symbols of arity > 0. Now, let Γ be such an ACUI-unification problem with constants and C and X be the finite sets of constants and variables respectively occurring in Γ. In order to check whether Γ has an ACUI-unifier or not, it is sufficient to consider substitutions that are the identity on $V \setminus X$ and replace every $x \in X$ by a term in $T_\Sigma(C)$, i.e., a ground term containing (in addition to $\mathbf{0}$) only constants from C. In fact, any ACUI-unifier of Γ can be turned into one satisfying these properties by making it the identity on $V \setminus X$ and replacing variables and constants in $F \setminus C$ occurring in $\sigma(x)$ for $x \in X$ with $\mathbf{0}$. If we apply such a substitution σ to the terms occurring in Γ, then we obtain terms in $T_\Sigma(C)$. According to Lemma 2.1, σ is an ACUI-unifier of Γ iff $S(\sigma(s)) = S(\sigma(t))$ for every equation $s \approx^? t \in \Gamma$, i.e., every constant $c \in C$ occurring on the left-hand side $\sigma(s)$ also occurs on the right-hand side $\sigma(t)$ and vice versa.

Example 2.3. *Consider the following* ACUI*-unification problem with constants:*

$$\begin{aligned}\Gamma = \{x_1 + x_2 &\approx^? a + b + c,\\ a + x_2 &\approx^? b + x_3,\\ x_1 + x_3 &\approx^? a + c\}\end{aligned}$$

We have $C = \{a, b, c\}$ and $X = \{x_1, x_2, x_3\}$. If we define $\sigma(x_1) := a + c$ and $\sigma(x_2) := b$, then σ solves the first equation, since then all the constants in C occur on both sides. In order to solve the second equation as well, we then must define $\sigma(x_3)$ such that a occurs in it, and c does not occcur in it. This is satisfied by setting $\sigma(x_3) := a$, which then also satisfies the third equation. Note that setting $\sigma(x_3) := a + b$ would have satisfied the second equation, but not the third. ♢

We will now recall Kapur and Narendran's [KN92] reduction of solvability of ACUI-unification problems to satisfiability of propositional Horn formulae. Each equation in the unification problem Γ is translated into several Horn clauses and the overall Horn formula is the conjunction of all clauses for all equations. In this reduction, we use propositional variables $p(a,x)$ for every $a \in C$ and $x \in X$. The intuitive semantics of these variables is that $p(a,x)$ is true iff a is *not* in $S(\sigma(x))$ for the given substitution σ.

It is easy to see that each equation $s \approx^? t \in \Gamma$ can be written in the form

$$s_0 + x_1 + \ldots + x_m \approx^? t_0 + y_1 + \ldots + y_n, \tag{2.1}$$

where $s_0, t_0 \in T_\Sigma(F)$, $m, n \geq 0$, $x_1, \ldots, x_m \in X$ are distinct variables, and $y_1, \ldots, y_n \in X$ are distinct variables.

Now, for each equation of the form (2.1) and each $a \in S(s_0) \setminus S(t_0)$ we generate the Horn clause

$$p(a, y_1) \wedge \ldots \wedge p(a, y_n) \rightarrow \bot.$$

Indeed, whenever an element $a \in C$ is in $S(s_0)$ but not in $S(t_0)$, for the equation to hold true, a must occur in the image of some y_j. The symmetric Horn clauses are also produced, i.e., for each $a \in S(t_0) \setminus S(s_0)$

$$p(a, x_1) \wedge \ldots \wedge p(a, x_m) \rightarrow \bot.$$

Constants $a \in S(s_0) \cap S(t_0)$ are obviously harmless since they automatically occur on both sides of the equation. Thus, it remains to deal with the constants $a \in C$ that are *not* in $S(s_0) \cup S(t_0)$. First, if such a constant a does not occur in the image of any of the variables on the right-hand side, then it should not occur in the image of any of the variables on the left-hand side, which is expressed by the Horn clauses

$$p(a, y_1) \wedge \ldots \wedge p(a, y_n) \rightarrow p(a, x_j) \qquad \text{for all } j = 1, \ldots, m.$$

Again, we also need the symmetric clauses, i.e., for each $a \notin C \setminus (S(s_0) \cup S(t_0))$ we produce

$$p(a, x_1) \wedge \ldots \wedge p(a, x_m) \rightarrow p(a, y_j) \qquad \text{for all } j = 1, \ldots, n.$$

It is easy to see that the Horn formula obtained by conjoining all the Horn clauses derived from the unification problem Γ is satisfiable iff Γ has a solution (see [KN92] for details). The number of derived Horn clauses and their sizes are obviously polynomial in the size of the given ACUI-unification problem Γ. Since satisfiability of propositional Horn formulae can be tested in linear time [DG84], this yields the following upper bound for deciding solvability of ACUI-unification problems with constants.

Proposition 2.4 ([KN92]). ACUI-*unification with constants is decidable in polynomial time.*$\diamondsuit$

2.3 Approximate unification modulo ACUI

Intuitively, in approximate unification we consider the case where a given unification problem is not solvable, and we ask what is the "best we can do" towards solving it. De facto, we will also consider solvable unification problems as input for approximate unification, but then producing an exact unifier should be the best we can do. Before introducing formal approaches for how to rank the quality of approximate unifiers, we illustrate the underlying ideas using an example.

Example 2.5. *Consider the following* ACUI-*unification problem with constants:*

$$\begin{aligned} \Gamma = \{x_1 + x_2 &=^? a, \\ x_1 + x_3 &=^? c, \\ b + c + x_2 &=^? b + x_3, \\ b + x_2 + x_3 &=^? c + x_1\}. \end{aligned}$$

Trying to solve the first two equations already fully determines the substitution $\sigma = \{x_1 \mapsto \mathbf{0}, x_2 \mapsto a, x_3 \mapsto c\}$. *Even though* σ *solves the first two equations, it does not*

solve the remaining two. Under σ, the set of free constants occurring in the left-hand side of the third equation becomes $\{a, b, c\}$, while the right-hand side yields the set $\{b, c\}$. Likewise for the fourth equation we obtain $\{a, b, c\}$ on the left- and $\{c\}$ on the right-hand side.

How far away is this substitution from being an exact solution? One idea is to count the number of equations that are not satisfied by it. In our example, this would yield the number 2 since σ does not solve the third and fourth equation. It is easy to see that, w.r.t. this measure, σ is actually the best we can do. In fact, in any solution of the fourth equation, b must occur in the image of x_1, and thus this substitution violates at least the first two equations. In addition, we have already seen that a solution of the first two equations cannot satisfy the third and fourth equations. Thus, it is not possible to satisfy more than two of the four equations.

The above measure fails, however, to assess how far from being solved are the violated equations. Another possibility is to count how many of the elements of C take part in at least one violation, i.e., occur on one side but not on the other when the substitution has been applied. In our example, there are two such constants, namely a and b. In fact, after applying σ, both a and b occur on the left-hand side of the fourth equation, but not on the right-hand side. In contrast, c does not take part in any violation. Again, it is not hard to show that this is the best we can do w.r.t. this measure.

Still, this new measure ignores for how many equations a given element of C takes part in a violation. To take this aspect into account, we will also consider a measure that counts the overall number of violations, i.e., sums up the number of equations each element of C violates. Returning to our example, this would be 3 violations for the substitution σ: 2 for a, 1 for b, and 0 for c. For this measure, we can actually do better than σ. In fact, if we define $\theta(x_1) := \theta(x_2) := \mathbf{0}$ and $\theta(x_3) := c$, then a violates only the first equation, b violates only the fourth equation, and c violates no equation. Consequently, θ gets assigned the value 2, which is better than 3, which was the value for σ. ♢

In this example, and also in our general definitions, we consider only substitutions that are the identity on $V \setminus X$ and assign terms in $T_\Sigma(C)$ to the variables in X. The reason is that the assignment to variables in $V \setminus X$ has no influence on the images of the left- and right-hand sides of equations in Γ, and that free constants and variables in the images may only introduce additional violations, but cannot remove violations caused by constants in C.

In the following, we will investigate all three of the measures sketched in the example, and determine the computational complexity of the corresponding decision problems, i.e., given a unification problem Γ and a natural number ℓ, is there a substitution whose value w.r.t. the given measure is $\leq \ell$. It should be clear that these decision problems are in NP. Any substitution basically assigns subsets of C to the variables, and thus all such substitutions can be guessed in nondeterministic polynomial time. For a given substitution, the value assigned to it by the respective measure can obviously be computed in polynomial time. However, for the cases that actually are NP-complete, we also provide a reduction to Max-HSAT [JS87], which is known to be NP-complete. This allows us to use existing optimized solvers for Max-HSAT (see for example [MIM17]) to solve approximate ACUI-unification problems. NP-hardness will be shown by a reduction from Max-HSAT.

For later reference, we now define this problem formally.

Definition 2.6 (Max-HSAT).

Input: *a nonnegative integer ℓ and a formula $\varphi = \bigwedge_{i=1}^{n} C_i$ over a finite set of propositional variables P, where C_i is a* Horn clause*, i.e., of the form $p_1 \wedge \cdots \wedge p_m \to q$, with $p_1, \ldots, p_m$*

being propositional variables, and q either a propositional variable or $\bot$*. Note that* $m = 0$ *is possible, where the empty conjunction is interpreted as* $\top$*.*

Question: *does there exist a set of indices* $I \subseteq \{1, \ldots, n\}$ *of cardinality at least* ℓ*, and a valuation* $\nu : P \to \{0, 1\}$ *such that* $\nu(C_i) = 1$ *for all* $i \in I$*.*

We call this decision problem Max-HSAT*. For a given* ℓ*, Max-HSAT(*ℓ*) consists of all Horn formulae* $\varphi = \bigwedge_{i=1}^{n} C_i$ *for which there is a valuation that satisfies at least* ℓ *clauses* C_i*.* ♢

2.3.1 Minimizing the number of violated equations

As mentioned above, we consider only substitutions that are the identity on $V \setminus X$ and assign terms in $T_\Sigma(C)$ to the variables in X. We say that such a substitution σ *violates* an equation of the form (2.1) if

$$S(s_0 + \sigma(x_1) + \ldots + \sigma(x_m)) \neq S(t_0 + \sigma(y_1) + \ldots + \sigma(y_n)).$$

Definition 2.7 (MinVEq-ACUI**).** *Given an* ACUI*-unification problem with constants* Γ *and a nonnegative integer* ℓ*, we now ask whether there exists a substitution* σ *such that at most* ℓ *of the equations of the system are violated by* σ*. We call this decision problem* MinVEq-ACUI*. For a given* ℓ*, MinVEq-*ACUI *(*ℓ*) consists of all* ACUI*-unification problems with constants for which there is a substitution that violates at most* ℓ *equations of the system.* ♢

We will show that MinVEq-ACUI is NP-complete using reductions to and from Max-HSAT.

Reducing MinVEq-ACUI to Max-HSAT

For this purpose, we introduce new propositional variables $good(i)$, whose rôle is to determine whether the ith equation is to be satisfied or not. We conjoin $good(i)$ to the left-hand side of each of the Horn clauses derived from the ith equation, i.e., if the ith equation is of the form (2.1), then we generate the following Horn clauses:

- For each $a \in S(s_0) \setminus S(t_0)$: $good(i) \wedge p(a, y_1) \wedge \ldots \wedge p(a, y_n) \to \bot$;
- For each $a \in S(t_0) \setminus S(s_0)$: $good(i) \wedge p(a, x_1) \wedge \ldots \wedge p(a, x_m) \to \bot$;
- For each $a \notin S(s_0) \cup S(t_0)$:

$$\begin{aligned} good(i) \wedge p(a, y_1) \wedge \ldots \wedge p(a, y_n) &\to p(a, x_j) && \text{for all } j = 1, \ldots, m; \\ good(i) \wedge p(a, x_1) \wedge \ldots \wedge p(a, x_m) &\to p(a, y_j) && \text{for all } j = 1, \ldots, n. \end{aligned}$$

- Furthermore, we add the Horn clause $\top \to good(i)$.

If k' is the number of clauses generated by the original reduction (see Section 2.2) and k is the number of equations in the unification problem Γ, then we obtain $k' + k$ Horn clauses in this modified reduction. Let $\varphi_\Gamma = C_1 \wedge \cdots \wedge C_{k'+k}$ denote the Horn formula obtained by conjoining these Horn clauses.

Before proving soundness and completeness of this reduction, we illustrate the above construction with an example.

Example 2.8. *Consider the following* ACUI*-unification problem with constants, which is not solvable:*

$$\begin{aligned}\Gamma = \{x_1 + x_2 &=^? a,\\ b + c + x_2 &=^? b + x_3,\\ x_1 \ + x_3 &=^? c\}.\end{aligned}$$

Then the reduction yields the following the Horn clauses:

$$\begin{array}{ll}
good(1) \wedge p(a,x_1) \wedge p(a,x_2) \to \bot & good(3) \to p(a,x_1)\\
good(1) \to p(b,x_1) & good(3) \to p(a,x_3)\\
good(1) \to p(b,x_2) & good(3) \to p(b,x_1)\\
good(1) \to p(c,x_1) & good(3) \to p(b,x_3)\\
good(1) \to p(c,x_2) & good(3) \wedge p(c,x_1) \wedge p(c,x_3) \to \bot\\
good(2) \wedge p(a,x_2) \to p(a,x_3) & \top \to good(1)\\
good(2) \wedge p(a,x_3) \to p(a,x_2) & \top \to good(2)\\
good(2) \wedge p(c,x_3) \to \bot & \top \to good(3)
\end{array}$$

The system contains $k = 3$ equations and the original reduction would produce $k' = 13$ clauses. Thus, φ_Γ contains $k' + k = 16$ Horn clauses.

The valuation that sets $p(a,x_1)$, $p(a,x_2)$, $p(a,x_3)$, $p(c,x_3)$ and $good(3)$ to false and all other variables to true satisfies all but the last clause. This corresponds to the substitution σ with $\sigma(x_1) = \sigma(x_2) = a$ and $\sigma(x_3) = a + c$, which satisfies the first two equations and violates the third one. ♢

Intuitively, setting the propositional variable $good(i)$ to false "switches off" the Horn clauses induced by the ith equation in the original reduction. Consequently, the satisfaction of these clauses is no longer enforced, which means that the ith equation may be violated. By maximizing satisfaction of the clauses $\top \to good(i)$, we thus minimize the number of violated equations from Γ. More precisely, we can show the following lemma.

Lemma 2.9. *Let Γ be an* ACUI*-unification problem with constants consisting of k equations and generating k' clauses in the reduction introduced in Section 2.2. Then we have, for all $0 \le \ell \le k$:*

$$\Gamma \in \textit{MinVEq-}\mathsf{ACUI}(\ell) \ \textit{ iff } \ \varphi_\Gamma \in \textit{Max-HSAT}((k'+k)-\ell).$$

Proof. Let $\Gamma \in$ MinVEq-ACUI(ℓ). This means that there exists a substitution σ such that at most ℓ of the equations in Γ are violated by σ. We use σ to define a valuation of the propositional variables occurring in φ_Γ. For every $a \in C$ and every variable $x \in X$ we set $p(a,x)$ to true iff $a \notin S(\sigma(x))$. In addition, we set $good(i)$ to true iff σ does not violate the ith equation of Γ.

Now, suppose that the ith equation of Γ is not violated by σ. Then our valuation makes all clauses corresponding to the ith equation evaluate to true, and in addition it also satisfies $\top \to good(i)$. If the jth equation of Γ is violated, then $good(j)$ is false. Consequently, all clauses of φ_Γ corresponding to the jth equation evaluate to true, but $\top \to good(j)$ evaluates

to false. Summing up, the valuation induced by σ satisfies all the clauses of φ_Γ, with the exception of the clauses $\top \to good(j)$ if the jth equation is violated by σ. Since σ violates $\leq \ell$ equations and there are $k'+k$ clauses in φ_Γ, the valuation satisfies $\geq (k'+k)-\ell$ clauses, which shows that $\varphi_\Gamma \in$ Max-HSAT$((k'+k)-\ell)$.

For the opposite direction, let $\varphi_\Gamma \in$ Max-HSAT$((k'+k)-\ell)$. This means that there is a valuation ν and a set of indices $I \subseteq \{1,\ldots,k'+k\}$ with $|I| \geqslant k'+k-\ell$ such that $\nu(C_i)=1$ for all $i \in I$. Since $k \geq \ell$, we know that $\geq k-\ell$ of the clauses $\top \to good(i)$ must evaluate to true, i.e. $\geq k-\ell$ of the propositional variables $good(i)$ evaluate to true. If i is an index for which $\nu(good(i))=1$, then all the clauses produced by the reduction of Kapur and Narendran (see Section 2.2) for the ith equation evaluate to true. By the correctness of that reduction, the ACUI-unification problem consisting of the equations for which $\nu(good(i))=1$ has a solution. Thus, there is a substitution that solves $\geq k-\ell$ equations of Γ, i.e., violates $\leq \ell$ equations. This shows that $\Gamma \in$ MinVEq-ACUI(ℓ). □

Since Max-HSAT is in NP, this lemma implies that MinVEq-ACUI also belongs to NP.

Reducing Max-HSAT to MinVEq-ACUI

Consider the Horn formula $\varphi = C_1 \wedge \ldots \wedge C_k$, where C_i is a Horn clause for $i = 1,\ldots,k$. To construct a corresponding ACUI-unification problem with constants Γ_φ, it is sufficient to use a single free constant a, i.e., we will have $C = \{a\}$. For every propositional variable p appearing in φ, we introduce a variable x_p. Intuitively, a occurs in x_p iff p is set to false. Now, each Horn clause in φ yields the following equations:

- If C_i is of the form $p_1 \wedge \ldots \wedge p_n \to p$, then the corresponding equation is

  $$x_{p_1} + \ldots + x_{p_n} + x_p =^? x_{p_1} + \ldots + x_{p_n}.$$

  Obviously, this equation enforces that a cannot occur in x_p if it does not occur in any of the variables x_{p_i}.

- If C_i is of the form $p_1 \wedge \ldots \wedge p_n \to \bot$, then the corresponding equation is

  $$x_{p_1} + \ldots + x_{p_n} =^? a.$$

  This equation enforces that a must occur in one of the variables x_{p_i}.

- If C_i is of the form $\top \to p$, then the corresponding equation is

  $$x_p =^? \mathbf{0}.$$

  This equation ensures that a cannot occur in x_p.

The following example illustrates the construction of Γ_φ.

Example 2.10. *Consider the Horn formula*

$$\varphi = (p_1 \wedge p_2 \to p_3) \wedge (p_1 \wedge p_3 \to \bot) \wedge (p_2 \wedge p_3 \to \bot) \wedge (\top \to p_1) \wedge (\top \to p_2).$$

The corresponding ACUI *unification problem consists of the following equations:*

$$x_{p_1} + x_{p_2} + x_{p_3} =^? x_{p_1} + x_{p_2},$$
$$x_{p_1} + x_{p_3} =^? a, \quad x_{p_2} + x_{p_3} =^? a,$$
$$x_{p_1} =^? \mathbf{0}, \quad x_{p_2} =^? \mathbf{0}.$$

It is easy to see that φ is not satisfiable. The valuation ν that makes p_1 and p_2 true, and p_3 false satisfies all clauses except for the first one. Given the intuition that a occurs in x_{p_i} iff p_i is set to false, this valuation induces the following substitution σ:

$$\sigma := \{x_{p_1} \mapsto \mathbf{0}, x_{p_2} \mapsto \mathbf{0}, x_{p_3} \mapsto a\},$$

which solves all equations in the ACUI*-unification problem except for the first one.* ♢

More generally, we will show that there is a 1–1-relationship between valuations satisfying certain clauses and substitutions satisfying the corresponding equations. However, note that in Max-HSAT the number of satisfied clauses is maximized whereas in MinVEq-ACUI the number of violated equations is minimized.

Lemma 2.11. *Let $\varphi = C_1 \wedge \ldots \wedge C_k$ be a Horn formula and Γ_φ the corresponding* ACUI *unification problem. Then we have, for all $0 \leq \ell \leq k$:*

$$\varphi \in \textit{Max-HSAT}(\ell) \textit{ iff } \Gamma_\varphi \in \textit{MinVEq-}\mathsf{ACUI}(k-\ell).$$

Proof. Suppose that $\varphi \in \text{Max-HSAT}(\ell)$. This means that there exists a valuation ν and a set of indices $I \subseteq \{1,\ldots,k\}$, $|I| \geq \ell$ such that $\nu(C_i) = 1$ for every $i \in I$. Given such a valuation ν, we define the substitution σ as follows:

$$\sigma(x_{p_i}) := a \text{ if } \nu(p_i) = 0 \text{ and } \sigma(x_{p_i}) := \mathbf{0} \text{ if } \nu(p_i) = 1.$$

We show that, for every $i \in I$, the ith equation is solved by this substitution. Indeed, if the ith equation is of the form:

- $x_{p_1} + \ldots + x_{p_n} + x_p =^? x_{p_1} + \ldots + x_{p_n}$, then $C_i = p_1 \wedge \cdots \wedge p_n \to p$ evaluates to true under ν. This means that either $\nu(p_j) = 0$ for some $j \in I$ or $\nu(p) = 1$. In the first case, a then occurs on both sides of the equation, and thus the equation is solved. In the second case, $\sigma(x_p) = \mathbf{0}$, and again the equation is solved by σ.
- $x_{p_1} + \cdots + x_{p_n} =^? a$, then $C_i = p_1 \wedge \cdots \wedge p_n \to \bot$ evaluates to true under ν. This means that $\nu(p_j) = 0$ for some $j \in I$, and thus $\sigma(x_{p_j}) = a$ for some $j \in I$. Consequently, $\sigma(x_{p_1} + \cdots + x_{p_n})$ is a sum of as and $\mathbf{0}$s, which implies that the ith equation is solved by σ.
- $x_p =^? \mathbf{0}$, then $C_i = \top \to p$ evaluates to true under ν. This means that $\nu(p) = 1$, and thus $\sigma(x_p) = \mathbf{0}$. This shows that the ith equation is solved by σ.

Consequently, the substitution σ solves at least ℓ equations of Γ_φ, and thus violates at most $k-\ell$ equations, which implies $\Gamma_\varphi \in \text{MinVEq-ACUI}(k-\ell)$.

For the opposite direction, if $\Gamma_\varphi \in \text{MinVEq-ACUI}(k-\ell)$, then there is a substitution σ such that at least ℓ equations of Γ_φ are *not* violated. We can assume without loss of generality

that σ uses a as the only free constant. If we define $\nu(p) = 0$ iff a occurs in $\sigma(x_p)$, then we can show (in the same way as above) that ν satisfies at least ℓ clauses of φ, which implies $\varphi \in$ Max-HSAT(ℓ). □

Since Max-HSAT is NP-hard, this lemma implies that MinVEq-ACUI is also NP-hard. Put together, the two lemmas yield the exact complexity of MinVEq-ACUI.

Theorem 2.12. *MinVEq-*ACUI *is NP-complete. The NP-hardness result holds even for* ACUI*-unification problems with only one free constant.*

2.3.2 Minimizing the number of violating elements

For the second measure, instead of minimizing the number of violated equations, we will minimize the number of violating elements of C, where as before C is the set of free constants occurring in the unification problem.

Given a substitution σ, we say that $a \in C$ *violates* an equation of the form (2.1) *w.r.t.* σ if

$$a \in S(s_0 + \sigma(x_1) + \ldots + \sigma(x_m)) \,\Delta\, S(t_0 + \sigma(y_1) + \ldots + \sigma(y_n)),$$

where Δ denotes the symmetric difference of two sets. We say that $a \in C$ *violates* Γ *w.r.t.* σ if it violates at least one equation in Γ w.r.t. σ.

Definition 2.13 (MinVEl-ACUI**).** *Given an* ACUI*-unification problem with constants* Γ *and a nonnegative integer* ℓ*, we now ask whether there exists a substitution* σ *such that at most* ℓ *constants violate* Γ *w.r.t.* σ*. We call this decision problem* MinVEl-ACUI*. For a given* ℓ*, MinVEl-*ACUI *(*ℓ*) consists of all* ACUI*-unification problems* Γ *with constants for which there is a substitution w.r.t. which at most* ℓ *constants violate* Γ*.* ♢

In contrast to the problem MinVEq-ACUI considered in the previous section, MinVEl-ACUI can be solved in polynomial time. In order to show this, we use projections of equations to free constants. As noted earlier, any term $t \in T_\Sigma(F, V)$ can be written in the form $t = t_0 + x_1 + \ldots + x_n$, where $t_0 \in T_\Sigma(F)$ is a ground term and $x_1, \ldots, x_n$ are variables in V. Given a constant $a \in F$ the *projection* of such a term onto a is defined to be $t^a = \pi_a(t_0) + x_1 + \ldots + x_n$, where for a ground term $t_0 \in T_\Sigma(F)$ we set $\pi_a(t_0) = a$ if a occurs in t_0, and $\mathbf{0}$ otherwise. Then the projection of an equation $s =^? t$ to a is $s^a =^? t^a$, and the projection of an ACUI-unification problem with constants Γ to a, denoted by Γ^a, is the system of the projections of the equations in Γ to a. Finally, the projection of a ground substitution σ to a is the substitution $\sigma^a : V \to T_\Sigma(\{a\})$ defined as $\sigma^a(x) := \sigma(x)^a$.

Consider the unification problem Γ and the substitution σ introduced in Example 2.8. The constant a violates Γ w.r.t. σ, while b, c do not. For the elements of $C = \{a, b, c\}$, we obtain the following projections of Γ:

$$\begin{array}{rcl} \multicolumn{3}{c}{\underline{\Gamma^a}} \\ x_1 + x_2 & =^? & a \\ x_2 & =^? & x_3 \\ x_1 + x_3 & =^? & \mathbf{0} \end{array} \qquad \begin{array}{rcl} \multicolumn{3}{c}{\underline{\Gamma^b}} \\ x_1 + x_2 & =^? & \mathbf{0} \\ b + x_2 & =^? & b + x_3 \\ x_1 + x_3 & =^? & \mathbf{0} \end{array} \qquad \begin{array}{rcl} \multicolumn{3}{c}{\underline{\Gamma^c}} \\ x_1 + x_2 & =^? & \mathbf{0} \\ c + x_2 & =^? & x_3 \\ x_1 + x_3 & =^? & c \end{array}$$

Likewise, for the substitution σ we obtain the projections:

$$\begin{array}{rcl|rcl|rcl} & \underline{\sigma^a} & & & \underline{\sigma^b} & & & \underline{\sigma^c} & \\ \sigma^a(x_1) & = & a & \sigma^b(x_1) & = & \mathbf{0} & \sigma^c(x_1) & = & \mathbf{0} \\ \sigma^a(x_2) & = & a & \sigma^b(x_2) & = & \mathbf{0} & \sigma^c(x_2) & = & \mathbf{0} \\ \sigma^a(x_3) & = & a & \sigma^b(x_3) & = & \mathbf{0} & \sigma^c(x_3) & = & c \end{array}$$

One can easily check that σ^b and σ^c solve Γ^b and Γ^c, respectively, whereas σ^a does not solve Γ^a. This is closely related to the fact that a violates Γ w.r.t. σ, but b and c do not.

Lemma 2.14. *Let Γ be an* ACUI*-unification problem with constants. Then the following holds:*

1. *The constant $a \in C$ violates Γ w.r.t. σ iff σ^a does not solve Γ^a.*
2. *Given substitutions $\sigma_a : V \to T_\Sigma(\{a\})$ for all $a \in C$, define the substitution $\sigma : V \to T_\Sigma(C)$ as*
$$\sigma(x) = \sum_{a \in C} \sigma_a(x) \text{ for all } x \in V.$$
Then we have $\sigma^a = \sigma_a$ for all $a \in C$.
3. *There is a substitution $\sigma : V \to T_\Sigma(C)$ such that at most ℓ of the elements of C violate Γ w.r.t. σ iff at most ℓ of the* ACUI*-unification problems Γ^a $(a \in C)$ are not solvable.*

Proof. We will show the first two facts stated in the lemma, and then use them to prove the third.

1. This is an easy consequence of the following equivalences. The constant a violates the equation $s =^? t \in \Gamma$ w.r.t. σ iff $a \in S(\sigma(s)) \Delta S(\sigma(t))$ iff $a \in S(\sigma(s)^a) \Delta S(\sigma(t)^a)$ iff $a \in S(\sigma^a(s^a)) \Delta S(\sigma^a(t^a))$ iff σ^a does not solve $s^a =^? t^a \in \Gamma^a$.
2. $\sigma^a(x) = (\sum_{a' \in C} \sigma'_a(x))^a = \sum_{a' \in C} (\sigma'_a(x)^a) = \sigma_a(x)^a = \sigma_a(x)$.
3. Suppose that there is a substitution $\sigma : V \to T_\Sigma(C)$ such that at most ℓ elements of C violate Γ w.r.t. σ. Because of the first fact this implies that at least $k - \ell$ of the projected unification problems Γ^a are solved by the projected substitutions σ^a. Consequently, at most ℓ of the projected problems Γ^a are not solvable.
   For the opposite direction, suppose that at most ℓ of the systems Γ^a are not solvable. For every $a \in C$, if Γ^a is solvable, let $\sigma_a : V \to T_\Sigma(\{a\})$ be a substitution that solves it, and an arbitrary substitution $V \to T_\Sigma(\{a\})$ otherwise. Define the substitution $\sigma : V \to T_\Sigma(C)$ as $\sigma(x) := \sum_{a \in C} \sigma_a(x)$ for all $x \in V$. Then the constant $a \in C$ violates Γ w.r.t. σ iff $\sigma^a = \sigma_a$ solves Γ^a iff Γ^a is solvable. Consequently, at most ℓ of the elements of C violate Γ w.r.t. σ.

This completes the proof of the lemma. □

Due to the third fact stated the above lemma, to check whether $\Gamma \in$ MinVEl-ACUI(ℓ), it is sufficient to check which of the ACUI-unification problems Γ^a for $a \in C$ are solvable. This can obviously be done in polynomial time.

Theorem 2.15. *The problem MinVEl-*ACUI *is in P.*

2.3.3 Minimizing the number of violations

A disadvantage of the measure used in the previous section is that it does not distinguish between constants that violate only one equation and those violating many equations. To overcome this problem, we count for each violating constant how many equations it actually violates. We say that $a \in C$ *violates* Γ p *times w.r.t.* σ if it violates p equations in Γ w.r.t. σ. Further, we say that σ *violates* Γ q *times* if $q = \sum_{a \in C} p_a$ where, for each $a \in C$, the element a violates Γ p_a times w.r.t. σ.

Definition 2.16 (MinV-ACUI). *Given an* ACUI*-unification problem with constants* Γ *and a nonnegative integer* ℓ*, we now ask whether there exists a substitution* σ *that violates* Γ *at most* ℓ *times. We call this decision problem* MinV-ACUI*. For a given threshold value* ℓ*, MinV-*ACUI *(*ℓ*) consists of those* ACUI*-unification problems with constants* Γ *for which there is a substitution that violates* Γ *at most* ℓ *times.* ♢

The approach used in Section 2.3.1 to solve MinVEq-ACUI can easily be adapted to solve this new problem. Basically, we now introduce propositional variables $good(i, a)$ (instead of simply $good(i)$) to characterize whether the element $a \in C$ violates the ith equation. We conjoin $good(i, a)$ to the left-hand side of each of the Horn clauses derived from the ith equation for a. Furthermore, we add the Horn clauses $\top \to good(i, a)$ instead of $\top \to good(i)$. Following the earlier notation, we obtain $k' + k|C|$ Horn clauses in this modified reduction, and again use φ_Γ to denote the Horn formula obtained this way.

If we apply this modified reduction to the ACUI-unification problems of Example 2.8, then we obtain the Horn clauses depicted in Fig. 2.17. Consider the substitution θ with $\theta(x_1) = \theta(x_2) = a$ and $\theta(x_3) = c$. Then a violates the second and the third equation, whereas b and c do not violate any equation w.r.t. θ. We can use θ to define a valuation ν, which sets $p(a, x_1)$, $p(a, x_2)$, $p(c, x_3)$, $good(2, a)$, $good(3, a)$ to false and all other propositional variables to true. This valuation satisfies all clauses in Fig. 2.17, except for $\top \to good(2, a)$ and $\top \to good(3, a)$.

The following lemma states correctness of the modified reduction. Since its proof is very similar to the proof of Lemma 2.9, we leave it to the reader.

Lemma 2.18. *Let* Γ *be an* ACUI*-unification problem consisting of* k *equations, and generating* k' *clauses in the reduction introduced in Section 2.2, and let* C *be the set of free constants occurring in* Γ*. Denote with* $\varphi_\Gamma = C_1 \wedge \cdots \wedge C_{k'+k|C|}$ *the Horn formula obtained by applying the the modified reduction to* Γ*. Then we have*

$$\Gamma \in \textit{MinV-}\mathsf{ACUI}(\ell) \;\; \textit{iff} \;\; \varphi_\Gamma \in \textit{Max-HSAT}((k' + k|C|) - \ell).$$

Since Max-HSAT is in NP, this lemma implies that MinV-ACUI is also in NP. NP-hardness of MinV-ACUI actually follows directly from Lemma 2.11. In fact, the reduction considered in this lemma requires only a single constant a. In this setting, counting the number of violated equations is the same as counting the number of all violations, and thus MinV-ACUI coincides with MinVEq-ACUI. This shows that MinV-ACUI is NP-hard.

Theorem 2.19. *The problem MinV-*ACUI *is NP-complete.*

$$good(1,a) \wedge p(a,x_1) \wedge p(a,x_2) \rightarrow \bot$$

$$good(1,b) \rightarrow p(b,x_1) \qquad good(3,a) \rightarrow p(a,x_1)$$
$$good(1,b) \rightarrow p(b,x_2) \qquad good(3,a) \rightarrow p(a,x_3)$$
$$good(1,c) \rightarrow p(c,x_1) \qquad good(3,b) \rightarrow p(b,x_1)$$
$$good(1,c) \rightarrow p(c,x_2) \qquad good(3,b) \rightarrow p(b,x_3)$$
$$good(2,a) \wedge p(a,x_2) \rightarrow p(a,x_3) \qquad good(3,c) \wedge p(c,x_1) \wedge p(c,x_3) \rightarrow \bot$$
$$good(2,a) \wedge p(a,x_3) \rightarrow p(a,x_2)$$
$$good(2,c) \wedge p(c,x_3) \rightarrow \bot$$

$$\top \rightarrow good(1,a) \qquad \top \rightarrow good(2,a) \qquad \top \rightarrow good(3,a)$$
$$\top \rightarrow good(1,b) \qquad \top \rightarrow good(2,b) \qquad \top \rightarrow good(3,b)$$
$$\top \rightarrow good(1,c) \qquad \top \rightarrow good(2,c) \qquad \top \rightarrow good(3,c)$$

Figure 2.17: The Horn clauses obtained by applying the modified reduction to Γ from Example 2.8.

2.4 Unification modulo ACUIG

In this section, we consider unification modulo ACUIG, i.e., ACUI extended with a finite set of ground identities G. We will prove that, in this setting, we obtain the same complexity bounds as for ACUI-unification. Initially, we will demonstrate that the word problem in ACUIG is decidable in polynomial time. Subsequently, we will use this result to prove that ACUIG-unification with constant restrictions, a notion that generalizes unification with constants, is decidable in P. Finally, using previous combination results from Baader and Schulz [BS93] and the hardness result for general ACUI-unification by Kapur and Narendran [KN92], we will conclude that general ACUIG-unification is NP-complete.

2.4.1 The word problem for ACUIG

Just as in Section 2.2, we consider the signature $\Sigma = \{+, \mathbf{0}\}$ and the equational theory ACUI that states that $+$ is an associative, commutative, and idempotent binary function symbol with unit $\mathbf{0}$. But now we extend ACUI with a finite set of ground identities $G \subseteq T_\Sigma(F) \times T_\Sigma(F)$, and denote the equational theory obtained this way by ACUIG. The *word problem* for ACUIG asks whether two given terms $s, t \in T_\Sigma(F)$ are equivalent modulo ACUIG, i.e., whether $s =_{\mathsf{ACUIG}} t$ holds or not. As already mentioned in the introduction, we can measure the complexity of this problem in two different ways. On the one hand, we can assume that G is fixed beforehand, and then consider the word problem for the fixed equational theory $\mathsf{ACUIG} = \mathsf{ACUI} \cup G$. The complexity of the word problem is then measured in the size of the input terms s, t. On the other hand, we can view G to be part of the input and then measure the complexity in the combined size of s, t, and G. If the complexity is measured in

terms of s, t only, we will call this *term complexity*, and otherwise *combined complexity*.[4] We will actually show that the word problem for ACUIG is in P for combined complexity, which obviously implies that it is also in P for term complexity.

Recall that modulo ACUI, two ground terms $s, t \in T_\Sigma(F)$ are equivalent iff they contain the same constants. However, in the presence of ground identities G, the latter condition is sufficient, but not necessary for two terms to be equivalent. In fact, $\mathsf{ACUI} \subseteq \mathsf{ACUIG}$ obviously yields that $S(s) = S(t)$ implies $s \approx_{\mathsf{ACUIG}} t$. However, the opposite direction need no longer hold, as shown by the following example. Consider the terms $s = b + a + a$ and $t = a + b + c$, with corresponding sets $S(s) = \{a, b\}$ and $S(t) = \{a, b, c\}$, and the ground theory $G = \{a + b \approx c\}$. We have $s = b + a + a \approx_{\mathsf{ACUI}} a + b + a + b \approx_G a + b + c = t$, and thus $s \approx_{\mathsf{ACUIG}} t$, even though $S(s) \neq S(t)$. Intuitively, the identity in G can be used to add c to the set $\{a, b\}$.

We will now show how to decide whether two terms are equivalent modulo ACUIG. For this purpose we saturate the sets of constants occurring in the terms using the identities in G to add constants, as we have done with c in our example.

Definition 2.20. *Given a finite set of constants $A \subseteq F$, its* saturation *A^G is obtained by iteratively applying the identities of G as follows:*

- *begin with setting $A^G := A$;*
- *as long as there is an identity $g_i \approx h_i$ in G such that $S(g_i) \subseteq A^G$ and $S(h_i) \not\subseteq A^G$ (or $S(h_i) \subseteq A^G$ and $S(g_i) \not\subseteq A^G$), extend A^G by setting $A^G := A^G \cup S(h_i)$ (respectively, by setting $A^G := A^G \cup S(g_i)$).* ♢

This saturation process terminates after a number of iterations that is bounded by the cardinality of G. In fact, once an identity $g_i \approx h_i$ is applied in the saturation process, it is no longer applicable since the set A^G then contains $S(g_i) \cup S(h_i)$. It is also easy to see that the result of the saturation does not depend on the order in which rules are applied. Thus, each finite set $A \subseteq F$ has a unique saturation A^G, which can be computed in time polynomial in the cardinality of A and the size of G.

Example 2.21. *Consider the set of ground identities*

$$G = \{a + b + c \approx d,\ b + c + e \approx f\}$$

and the term $s = a + f$, which yields the start set $A = \{a, f\}$. The saturation process for A starts with setting $A^G := \{a, f\}$. For the second identity, we have that $S(f) = \{f\} \subseteq A^G$, but $S(b + c + e) = \{b, c, e\} \not\subseteq A^G$. Hence, we can extend A^G to the new set $A^G := A^G \cup S(b + c + e) = \{a, b, c, e, f\}$. Now, for the first identity, we have that $S(a + b + c) = \{a, b, c\} \subseteq A^G$, but $S(d) = \{d\} \not\subseteq A^G$, and thus we obtain $A^G := A^G \cup S(d) = \{a, b, c, d, e, f\}$. This is the final saturated set since it cannot be further extended using the identities in G. ♢

The following lemma is an easy consequence of the definition of saturation.

[4] Note that the word problem when the equational theory is part of the input is usually called the *uniform word problem* [BHS87].

Lemma 2.22. *Let A, B be finite subsets of F. Then the following holds:*

$$A \subseteq A^G,\ \ (A^G)^G = A^G,\ \ A \subseteq B \Rightarrow A^G \subseteq B^G,\ \ A^G \cup B^G \subseteq (A \cup B)^G.$$

Proposition 2.23. *Let $s, t \in T_\Sigma(F)$. Then $s \approx_{\mathsf{ACUIG}} t$ iff $S(s)^G = S(t)^G$. In particular, the combined and thus also the term complexity for the word problem for* ACUIG *is in P.* ♢

Proof. Decidability in polynomial time obviously follows from the equivalence in the first statement since the saturation A^G of a finite set $A \subseteq F$ can be computed in polynomial time, and the cardinality of $S(s), S(t)$ is bounded by the size of s, t.

To show the equivalence, first assume that $S(s)^G = S(t)^G$. To conclude from this that $s \approx_{\mathsf{ACUIG}} t$, it is sufficient to show that saturation steps correspond to rewrite steps in ACUIG. Thus, assume that $l \in T_\Sigma(F)$, and that $g_i \approx h_i$ is an identity in G such that $S(g_i) \subseteq S(l)$. Then l is of the form $l \approx_{\mathsf{ACUI}} g_i + l'$. We now have $l \approx_{\mathsf{ACUI}} g_i + l' \approx_{\mathsf{ACUI}} g_i + g_i + l' \approx_{\mathsf{ACUIG}} h_i + l$, and $S(h_i + l) = S(l) \cup S(h_i)$. This shows that there are terms $s^G, t^G \in T_\Sigma(F)$ such that $u \approx_{\mathsf{ACUIG}} u^G$ and $S(u^G) = S(u)^G$ for $u \in \{s, t\}$. By Lemma 2.1 we thus know that $S(s)^G = S(t)^G$ implies $s^G =_{\mathsf{ACUI}} t^G$, and thus we have $s \approx_{\mathsf{ACUIG}} s^G \approx_{\mathsf{ACUI}} t^G \approx_{\mathsf{ACUIG}} t$.

Second, assume that $S(s)^G \neq S(t)^G$. To show that this implies $s \not\approx_{\mathsf{ACUIG}} t$, we construct a model $\mathcal{A}$ of ACUIG in which the identity $s \approx t$ does not hold. As interpretation domain, we use all saturated sets over the constants occurring in s, t, G, i.e., $\Delta := \{A^G \mid A \subseteq C\}$ where C consists of the elements of F that occur in s, t, or G. Since saturation adds only constants occurring in G, we know that $A \subseteq C$ implies $A^G \subseteq C$. The binary symbol $+$ is interpreted as union followed by saturation, i.e., $A^G + B^G := (A^G \cup B^G)^G$, $\mathbf{0}$ as $\emptyset^G$, and $c \in C$ as $\{c\}^G$. Given a term $u \in T_\Sigma(C)$, its interpretation in this algebra is $S(u)^G$. This can easily be shown by induction on the structure of u, where the induction step uses the fact that

$$(A^G \cup B^G)^G = (A \cup B)^G, \tag{2.2}$$

which is an easy consequence of Lemma 2.22. Thus, $S(s)^G \neq S(t)^G$ implies that the terms s, t have different interpretations in $\mathcal{A}$. To show $s \not\approx_{\mathsf{ACUIG}} t$, it is thus sufficient to show that $\mathcal{A}$ satisfies all identities of ACUIG. For the identities in ACUI this is an easy consequence of (2.2) and the fact that set union is associative, commutative and idempotent and has $\emptyset$ as unit. Now consider an identity $g_i \approx h_i \in G$. When saturating the corresponding sets $S(g_i)$ and $S(h_i)$, one can in a first step go both from $S(g_i)$ and from $S(h_i)$ to $S(g_i) \cup S(h_i)$ (unless this step is void due to an inclusion). Saturating further, one thus obtains identical saturated sets, which shows that g_i and h_i are interpreted in $\mathcal{A}$ by the same saturated set. □

Continuing Example 2.21, recall that the term $s = a + f$ has the saturated set $S(s)^G = \{a, b, c, d, e, f\}$. It is easy to see that, for $t = b + d + e$, saturation produces the sequence of sets

$$S(t) = \{b, d, e\} \to \{a, b, c, d, e\} \to \{a, b, c, d, e, f\} = S(t)^G,$$

where in the first step the identity $d \approx a + b + c$ is applied, and in the second the identity $b + c + e \approx f$. Thus, we have $S(s)^G = S(t)^G$, which shows that $s = a + f \approx_{\mathsf{ACUIG}} b + d + e = t$.

2.4.2 ACUIG-unification with constant restriction

As in the previous subsection, let $\Sigma = \{+, \mathbf{0}\}$, F a countably infinite set of constants, and V a countably infinite set of variables. Given a finite set of ground identities $G \subseteq T_\Sigma(F) \times T_\Sigma(F)$, we now consider unification modulo $\mathsf{ACUIG} = \mathsf{ACUI} \cup G$. Note that in this setting the constants occurring in G are no longer free constants, but theory constants. Thus, an ACUIG-unification problem with constants may contain the constant $\mathbf{0}$, the constants from G, and additional free constants, i.e., elements of F that do not occur in G. For a given unification problem, a constant restriction prohibits the occurrence of certain free constants in the image of certain variables.

Definition 2.24 (ACUIG-unification problem with constant restriction).

Input: *A finite system $\Gamma = \{s_1 \approx^? t_1, \ldots, s_k \approx^? t_k\}$ of equations between terms in $T_\Sigma(V, F)$, a finite set of ground identities $G = \{g_1 \approx h_1, \ldots, g_m \approx h_m\}$ between terms in $T_\Sigma(F)$, and a mapping $\mathfrak{r} : D \to 2^X$ where $X \subseteq V$ is the set of variables occurring in Γ and $D \subseteq F$ is the set of free constants occurring in Γ.*

Question: *Is there a substitution σ such that $\sigma(s_i) \approx_{\mathsf{ACUIG}} \sigma(t_i)$ for every $i = 1, \ldots, k$ and for every $x \in X$ and $d \in D$ we have that d does not occur in $\sigma(x)$ if $x \in \mathfrak{r}(d)$. Such a substitution is called an ACUIG-unifier of Γ w.r.t. $\mathfrak{r}$.* ♢

Note that we consider the set of ground identities G to be part of the input. Thus, the complexity upper bound of P shown below for deciding ACUIG-unification with constant restriction holds for combined complexity, which implies the same upper bound also for term complexity.

In the following, we assume that $C \subseteq F$ is the set of constants occurring in Γ or G, and $X \subseteq V$ is the set of variables occurring in Γ. As before, in order to check whether Γ has a unifier w.r.t. $\mathfrak{r}$, it is sufficient to consider substitutions that are the identity on $V \setminus X$ and replace every $x \in X$ by a term in $T_\Sigma(C)$. In fact, any ACUIG-unifier of Γ w.r.t. $\mathfrak{r}$ can be turned into one satisfying these properties by making it the identity on $V \setminus X$ and replacing variables and constants in $F \setminus C$ occurring in $\sigma(x)$ for $x \in X$ with $\mathbf{0}$.

Intuitively, our algorithm for solving ACUIG-unification with constant restriction starts with a maximal substitution σ that respects the constant restriction, in the sense that $\sigma(x)$ for $x \in X$ contains as many of the constants from C as admitted by the restriction. Next, whenever an equation is not satisfied, that is, when a constant appears on one side but not on the other after saturation, we trim the substitution, so that it no longer introduces this violation. Upon termination, the algorithm provides a solution if one exists, and outputs **Fail** otherwise. In the algorithm we will use the following notation for turning a set of constants into the term that sums up theses constants: given $A \subseteq C$, we denote with $\sum(A)$ the term $\sum_{a \in A} a$. Note that $S(\sum(A)) = A$ and $\sum(S(s)) \approx_{\mathsf{ACUI}} s$ for all $s \in T_\Sigma(C)$.

Before proving correctness of this algorithm, we illustrate how it works on two examples, one where the input is a solvable unification problem, and one where the input problem is not solvable.

Example 2.25. *Consider the system of equations*

$$\Gamma = \{g + x_2 \approx^? a + x_1,\ b + x_1 \approx^? c + f + g,\ c + x_2 \approx^? a + c + e\},$$

Algorithm 1: ACUIG-unification with constant restriction

Input: An ACUIG-unification problem with constant restriction $\mathfrak{r}$, as introduced in Definition 2.24.
Output: A unifier σ w.r.t. $\mathfrak{r}$ or **Fail**.

Set $\sigma(x) := \sum(\{c \in C \mid x \notin \mathfrak{r}(c) \text{ for } c \in D \text{ or } c \text{ occurs in } G\})$ for all $x \in X$
while *some equation* $s \approx^? t$ *in* Γ *is not satisfied by* σ **do**
  if *there is a variable* x *in* s *such that* $S(\sigma(x)) \not\subseteq S(\sigma(t))^G$ *or* y *in* t *such that* $S(\sigma(y)) \not\subseteq S(\sigma(s))^G$ **then**
    Set $\sigma(x) := \sum(S(\sigma(x)) \cap S(\sigma(t))^G)$ for all variables x in s
    Set $\sigma(y) := \sum(S(\sigma(y)) \cap S(\sigma(s))^G)$ for all variables y in t
  else
    return Fail
  end
end
return σ

the set of ground identities $G = \{a + b + c \approx d,\ b + c + e \approx f\}$ *from Example 2.21, and the constant restriction*

$$\mathfrak{r}(g) = \{x_2\}.$$

Note that g *is the only* free *constant occurring in* Γ*, and thus it is the only constant occurring in this constant restriction. Also note that, if we had* $x_1 \in \mathfrak{r}(g)$*, then the second equation of* Γ *would not be solvable. In addition, without G, this second equation would not be solvable either:* b *belongs to the left-hand side, but could never belong to the right-hand side without additional ground identities.*

The algorithm begins by setting

$$\sigma(x_1) := a + b + c + d + e + f + g \ \text{ and } \ \sigma(x_2) := a + b + c + d + e + f.$$

Next, the algorithm enters the while loop and picks in each iteration an equation that is not satisfied:

- *The second equation is not satisfied by* σ*. In fact,* $S(\sigma(c + f + g))^G = \{b, c, e, f, g\}$*, and hence* $S(\sigma(x_1)) \not\subseteq S(\sigma(c + f + g))^G$*. The algorithm then proceeds to set* $\sigma(x_1) := \sum(\{a, b, c, d, e, f, g\} \cap \{b, c, e, f, g\}) = b + c + e + f + g$*.*
- *The third equation is not satisfied by* σ*. In fact,* $S(\sigma(a + c + e))^G = \{a, c, e\}$*, and hence* $S(\sigma(x_2)) \not\subseteq S(\sigma(a + c + e))^G$*. The algorithm then proceeds to set* $\sigma(x_2) := \sum(\{a, b, c, d, e, f\} \cap \{a, c, e\}) = a + c + e$*.*
- *The first equation is not satisfied by* σ*. In fact, we have that* $S(\sigma(x_1)) = \{b, c, e, f, g\} \not\subseteq S(\sigma(g + x_2))^G = \{a, c, e, g\}$*. The algorithm proceeds to set* $\sigma(x_1) := \sum(\{b, c, e, f, g\} \cap \{a, c, e, g\}) = c + e + g$*.*

The algorithm then terminates since all equations are satisfied, and yields the substitution $\sigma = \{x_1 \mapsto c + e + g, x_2 \mapsto a + c + e\}$ *as output, which is indeed an* ACUIG*-unifier of* Γ *that respects the constant restriction* $\mathfrak{r}$*.* ♢

Next, we provide an instance that admits no solution, and demonstrate how the algorithm reaches this conclusion.

Example 2.26. *Consider the system of equations*

$$\Gamma = \{a + x_2 \approx^? x_1,\ b + x_1 \approx^? c + f + g,\ c + x_2 \approx^? c + e\},$$

the set of ground identities $G = \{a + b + c \approx d,\ b + c + e \approx f\}$ *considered in the previous example, and the constant restriction*

$$\mathfrak{r}(g) = \emptyset.$$

Note that g *is the only free constant occurring in* Γ*, and that* $\mathfrak{r}(g) = \emptyset$ *means that, in effect, there is no constant restriction, i.e., all substitutions are admissible w.r.t. this constant restriction.*

The algorithm begins by setting

$$\sigma(x_1) := a + b + c + d + e + f + g \text{ and } \sigma(x_2) := a + b + c + d + e + f + g.$$

Next, the algorithm enters the while loop and picks in each iteration an equation that is not satisfied:

- *The second equation is not satisfied by* σ*. In fact,* $S(\sigma(c + f + g))^G = \{b, c, e, f, g\}$*, and hence* $\sigma(x_1) \not\subseteq S(\sigma(c + f + g))^G$*. The algorithm then proceeds to set* $\sigma(x_1) := \sum(\{a, b, c, d, e, f, g\} \cap \{b, c, e, f, g\}) = b + c + e + f + g$*.*
- *The third equation is not satisfied by* σ*. In fact, we have that* $S(\sigma(c + e))^G = \{c, e\}$*, and hence* $\sigma(x_2) \not\subseteq S(\sigma(c + e))^G$*. The algorithm then proceeds to set* $\sigma(x_2) := \sum(\{a, b, c, d, e, f, g\} \cap \{c, e\}) = c + e$*.*
- *The first equation is not satisfied by* σ*. In fact, we have* $S(\sigma(x_1)) = \{b, c, e, f, g\} \not\subseteq S(\sigma(a + x_2))^G = \{a, c, e\}$*. The algorithm proceeds to set* $\sigma(x_1) := \sum(\{b, c, e, f, g\} \cap \{a, c, e\}) = c + e$ *and* $\sigma(x_2) := \sum(\{c, e\} \cap \{b, c, e, f, g\}) = c + e$*.*
- *The first equation is still not satisfied by* σ*. However, the* if *condition of the algorithm is not satisfied: we have* $S(\sigma(x_1)) = \{c, e\} \subseteq S(\sigma(a + x_2))^G = \{a, c, e\}$ *and* $S(\sigma(x_2)) = \{c, e\} \subseteq S(\sigma(x_1))^G = \{c, e\}$*. Hence the if condition is not satisfied and the algorithm returns* ***Fail****.*

It is not hard to see that the problem indeed does not have an ACUIG*-unifier. Basically, the first equation implies that, for any unifier* σ*,* a *or* d *must occur in* $\sigma(x_1)$ *since otherwise* a *cannot be produced on the right-hand side of the first equation. However, any such* σ *then violates the second equation.* ♢

Proposition 2.27. *Algorithm 1 terminates in time polynomial in the size of* G *and* Γ*. If* Γ *has an* ACUIG*-unifier w.r.t.* $\mathfrak{r}$*, then it provides such a unifier as output, and otherwise it fails.* ♢

Proof. Termination in polynomial time is an easy consequence of the facts that (i) in each iteration of the while-loop, at least one constant is removed from the image of a variable, or the loop is exited; and that (ii) saturation can be done in polynomial time.

Since the algorithm only returns a substitution if the while-loop is exited regularly, this substitution satisfies all the equations of Γ. It satisfies the constant restriction due to the

fact that the original substitution satisfies it and that constants are only removed from, but never added to, the image of variables during the run of the algorithm. Consequently, if the algorithm returns a substitution, then this substitution is an ACUIG-unifier of Γ w.r.t. $\mathfrak{r}$. This shows that the algorithm must return **Fail** in case Γ has no unifier w.r.t. $\mathfrak{r}$.

To prove the completeness of the algorithm, assume that $\widehat{\sigma}$ is a ACUIG-unifier of Γ w.r.t. $\mathfrak{r}$, and that the algorithm terminates during the rth iteration of the while-loop. Let $\sigma^{(0)}$ be the substitution σ before the first iteration of the while-loop. For $i \in \{1, \ldots, r-1\}$, let $\sigma^{(i)}$ be the substitution obtained at the end of the ith iteration of the while-loop.

We extend $\subseteq$ to substitutions in a natural way, by using point-wise comparison of constant sets, that is, $\tau \subseteq \tau'$ iff $S(\tau(x)) \subseteq S(\tau'(x))$ for all $x \in X$. We prove by induction on i that $\widehat{\sigma} \subseteq \sigma^{(i)}$, for all $i \in \{0, \ldots, r-1\}$. The base case $i = 0$ is obvious: since $\widehat{\sigma}$ satisfies the constant restriction, we clearly have $\widehat{\sigma} \subseteq \sigma^{(0)}$. Let now $i \in \{0, \ldots, r-2\}$, and assume that we already know that $\widehat{\sigma} \subseteq \sigma^{(i)}$. We must prove $\widehat{\sigma} \subseteq \sigma^{(i+1)}$.

Since the algorithm does not exit the while-loop at this iteration, there is an equation $s \approx^? t$ in Γ that is not satisfied by $\sigma^{(i)}$. In addition, since the algorithm does not fail at iteration i, there exists a variable x in s such that $S(\sigma(x)) \not\subseteq S(\sigma(t))^G$ or y in t such that $S(\sigma(y)) \not\subseteq S(\sigma(s))^G$. Clearly, for every $x \in X$ that does not appear in this equation, we have $S(\widehat{\sigma}(x)) \subseteq S(\sigma^{(i)}(x)) = S(\sigma^{(i+1)}(x))$. Let now x be a variable occurring in s (variables in t can be treated analogously). To prove that $S(\widehat{\sigma}(x)) \subseteq S(\sigma^{(i+1)}(x))$, it suffices to prove that $S(\widehat{\sigma}(x)) \subseteq S(\sigma^{(i)}(x))$ and that $S(\widehat{\sigma}(x)) \subseteq S(\sigma^{(i)}(t))^G$. The first statement is true by the induction hypothesis. Now, we have

$$S(\widehat{\sigma}(x)) \overset{(1)}{\subseteq} S(\widehat{\sigma}(s)) \overset{(2)}{\subseteq} S(\widehat{\sigma}(s))^G \overset{(3)}{=} S(\widehat{\sigma}(t))^G \overset{(4)}{\subseteq} S(\sigma^{(i)}(t))^G,$$

where (1) holds because x occurs in s, (2) by Lemma 2.22, (3) because $\widehat{\sigma}$ is a unifier of Γ, and (4) by Lemma 2.22 since $\widehat{\sigma} \subseteq \sigma^{(i)}$. This finishes the induction proof.

Therefore, we now know that $\widehat{\sigma} \subseteq \sigma^{(r-1)}$. There are two possible reasons for the algorithm terminating in the rth iteration. Either the while-loop is exited regularly or the algorithm returns **Fail**. In the first case, $\sigma^{(r-1)}$ is a unifier and the algorithm returns this substitution.

It remains to show that the second case cannot occur. In this case, we have that $S(\sigma^{(r-1)}(s))^G \neq S(\sigma^{(r-1)}(t))^G$ for some equation $s \approx^? t$ in Γ, but $S(\sigma^{(r-1)}(x)) \subseteq S(\sigma^{(r-1)}(t))^G$ for all variables x in s and $S(\sigma^{(r-1)}(y)) \subseteq S(\sigma^{(r-1)}(s))^G$ for all variables y in t. This can only be the case if there is a constant $c \in C$ such that c occurs in s, but $c \notin S(\sigma^{(r-1)}(t))^G$; or c occurs in t, but $c \notin S(\sigma^{(r-1)}(s))^G$. We show that this is impossible.

Thus assume that c occurs in s (the case where c occurs in t can be treated symmetrically). We have

$$c \overset{(1)}{\in} S(\widehat{\sigma}(s)) \overset{(2)}{\subseteq} S(\widehat{\sigma}(s))^G \overset{(3)}{=} S(\widehat{\sigma}(t))^G \overset{(4)}{\subseteq} S(\sigma^{(r-1)}(t))^G,$$

where (1) holds since c occurs in s, (2) by Lemma 2.22, (3) since $\widehat{\sigma}$ is a unifier of Γ, and (4) by Lemma 2.22 since $\widehat{\sigma} \subseteq \sigma^{(r-1)}$. □

The following theorem is an immediate consequence of Proposition 2.27.

Theorem 2.28. *The combined and thus also the term complexity of* ACUIG*-unification with constant restriction is in P.*

Note that this implies that elementary ACUIG-unification and ACUIG-unification with constants are also decidable in polynomial time. The next section deals with general ACUIG-unification.

2.4.3 General ACUIG-unification

General ACUIG-unification problems Γ differ from the ones we have considered until now in that the terms used in Γ may contain free function symbols, i.e., function symbols not occurring in the identities of ACUIG. For example, $\{f(x+a, a+b) \approx^? f(b+y, x)\}$ is such a general ACUIG-unification problem since it contains the additional function symbol f, which does not occur in the identities of ACUIG. Using techniques for the combination of unification algorithms, one can transfer complexity results for unification with constant restrictions to general unification. In fact, the following was proved by Baader and Schulz [BS96] for arbitrary equational theories E, where linear constant restrictions are a special form of constant restrictions.

Theorem 2.29 ([BS96]). *If solvability of E-unification problems with linear constant restriction is decidable in NP, then solvability of general E-unification problems is also decidable in NP.*

Together with our result from the previous subsection, this provides us with an NP-upper bound for general ACUIG-unification. The corresponding lower bound can be obtained by adapting the proof of NP-hardness for general ACUI-unification by Kapur and Narendran [KN92].

Theorem 2.30. *General* ACUIG*-unification is NP-complete, both w.r.t. combined complexity and w.r.t. term complexity.*

Proof. Membership in NP (for combined complexity, and thus also for term complexity) is an immediate consequence of Theorem 2.28 together with Theorem 2.29 since P $\subseteq$ NP and any linear constant restriction is a constant restriction.

We show that NP-hardness (w.r.t. term complexity) holds for any fixed finite set of ground identities G. This obviously implies NP-hardness also for combined complexity. This NP-hardness result can be shown by the same reduction from the set-matching problem used in [KN86] to show that general *ACI*-unification is NP-hard. To be more precise, this reduction yields *ACI*-unification problems of the form

$$\Gamma = \{g(s_1) + \ldots + g(s_m) \approx^? g(t_1) + \ldots + g(t_n)\},$$

where $+$ is an associative, commutative, and idempotent function symbol, g is a unary free function symbol and the terms $s_1, \ldots, s_m, t_1, \ldots, t_n$ contain only free function symbols and variables. The presence of a unit and of ground identities in ACUIG do not change solvability of such problems compared to *ACI* since

- in the top-level sum the additional identities cannot be used due to the fact that all terms on this level start with the free function symbol g;
- while the variables occurring in the terms $g(s_1), \ldots, g(s_m), g(t_1), \ldots, g(t_n)$ may be replaced by terms containing $+$ and constant symbols from G, these "alien" subterms can be abstracted away by free constants.

This shows that such a problem Γ is solvable modulo *ACI* iff it is solvable modulo ACUIG for an arbitrary finite set of ground identities G, which completes our proof of NP-hardness of general ACUIG-unification w.r.t. term complexity. □

2.5 Approximate Unification Modulo ACUIG

In this section we investigate how the addition of a ground theory to ACUI affects the complexity of approximate unification. Basically, we will investigate the same three measures that we treated in Section 2.3, but must adapt their definition to the extended setting.

First, one needs to decide which are the constants that can lead to violations. In Section 2.3, these were the free constants occurring in the problem since presence or absence of the theory constant $\mathbf{0}$ cannot contribute to the violation of an equation. For example, given the equation $x + a \approx^? a$, the substitution $\sigma = \{x \mapsto \mathbf{0}\}$ solves this equation, although syntactically, after applying this substitution, the left-hand side $\mathbf{0} + a$ contains $\mathbf{0}$ whereas the right-hand side a does not. This is the reason why we did not consider $\mathbf{0}$ when defining the set of constants $S(t)$ contained in a term t. In contrast, constants occurring in G may lead to a violation. For instance, consider the equation from above and the ground theory $G = \{b + c \approx b\}$. The substitution $\theta = \{x \mapsto b\}$ does not solve the equation since $\theta(x + a) = b + a \not\approx_{\mathsf{ACUIG}} a = \theta(a)$. Here, the constant b, which is a theory constant for ACUIG, violates the equation. This motivates to consider both the free constants occurring in Γ and the theory constants different from $\mathbf{0}$ occurring in G when counting violations. However, when defining under what conditions such a constant violates an equation, one cannot just consider the constants explicitly occurring on the left- and the right-hand side after applying the substitution. In fact, if we consider the equation $x + c \approx^? b$, then the substitution θ from above solves this equation, although c occurs in $\theta(x + c) = b + c$, but not in $\theta(b) = b$. To see whether a theory constant from G really violates an equation, we first need to apply saturation to the sets of constants occurring on each side of the equation.

In the following, let G be a ground theory, i.e., a finite set of ground identities between terms of $T_\Sigma(F)$, and $\Gamma = \{s_1 \approx^? t_1, \ldots, s_k \approx^? t_k\}$ an ACUIG-unification problem with constants. We denote the set of constants appearing in Γ or G with C and the set of variables appearing in Γ with X. As before, it is sufficient to consider only substitutions that are the identity on $V \setminus X$ and assign terms in $T_\Sigma(C)$ to variables in X.

Definition 2.31. *The substitution* σ violates *the equation* $s \approx^? t \in \Gamma$ modulo G *if*

$$S(\sigma(s))^G \neq S(\sigma(t))^G.$$

Furthermore, the constant $a \in C$ violates $s \approx^? t \in \Gamma$ modulo G w.r.t. σ *if*

$$a \in S(\sigma(s))^G \,\Delta\, S(\sigma(t))^G, \qquad \diamondsuit$$

and it violates Γ modulo G w.r.t. σ *if it violates at least one equation in* Γ *modulo* G *w.r.t.* σ.

Moreover, $a \in C$ violates Γ modulo G p times w.r.t. σ *if it violates* p *equations in* Γ *modulo* G *w.r.t.* σ. *Finally, we say that* σ violates Γ q times modulo G *if* $q = \sum_{a \in C} p_a$ *where, for each* $a \in C$, *the element* a *violates* Γ *modulo* G p_a *times w.r.t.* σ.

Given these redefined notions of violation, we can now formalize the three different decision problems for approximate ACUIG-unification analogously to how this was done in Section 2.3 for ACUI.

Definition 2.32. *Let G be a ground theory, i.e., a finite set of ground identities between terms of $T_\Sigma(F)$. Given an* ACUIG*-unification problem with constants Γ and a nonnegative integer ℓ, we ask whether there exists a substitution σ such that:*

- *at most ℓ of the equations of the system are violated modulo G by σ. We call this decision problem* MinVEq-ACUIG.
- *at most ℓ constants violate Γ modulo G w.r.t. σ. We call this decision problem* MinVEl-ACUIG.
- *σ violates Γ modulo G at most ℓ times. We call this decision problem* MinV-ACUIG. ♢

It is easy to see that all three problems belong to NP w.r.t. combined complexity, and thus also w.r.t. term complexity. In fact, as mentioned before, we can restrict the attention to substitutions that assign terms in $T_\Sigma(C)$ to the variables in X. For every $x \in X$, the assigned term $\sigma(x)$ is determined modulo ACUI by the set of constants $S(\sigma(x))$ occurring in it. Consequently, any such substitution can be guessed in nondeterministic polynomial time. For a given substitution, the saturations of the constant sets on the left- and right-hand sides of the equations can be computed in polynomial time due to Proposition 2.23, and thus the violations modulo G can be counted also in polynomial time. The only place where the ground theory plays a role in this straightforward NP-procedure is during the computation of the saturations. Since this computation is polynomial w.r.t. combined complexity, the NP upper bound also holds for combined complexity.

Proposition 2.33. *The problems MinVEq-*ACUIG*, MinVEl-*ACUIG*, and MinV-*ACUIG *are in NP w.r.t. combined complexity, and thus also w.r.t. term complexity.* ♢

In the next subsection, we will show that, for MinVEq-ACUIG and MinV-ACUIG, this NP upper bound is optimal, by showing NP-hardness w.r.t. term complexity for any fixed finite set of ground identities G. Note that the corresponding two problems are also NP-hard for ACUI, and actually MinVEq-ACUI and MinV-ACUI are used in the reduction that establishes these hardness results.

Analyzing the complexity of MinVEl-ACUIG turns out to be more subtle. Recall that the corresponding problem is in P for ACUI. We will show that this upper bound in general does not transfer from ACUI to ACUIG. In fact, we will exhibit a fixed finite ground theory G for which the problem MinVEl-ACUIG is NP-hard w.r.t. term complexity. Obviously, this implies that MinVEl-ACUIG is NP-hard w.r.t. combined complexity. However, there are fixed finite ground theories G for which MinVEl-ACUIG is in P w.r.t. term complexity. An obvious example is $G = \emptyset$ since we know that MinVEl-ACUI is in P, but we will also give some other examples of such theories.

2.5.1 The problems MinVEq-ACUIG and MinV-ACUIG

We show NP-hardness of MinVEq-ACUIG and MinV-ACUIG by reduction from MinVEq-ACUI and MinV-ACUI, respectively.

Lemma 2.34. *Let G be a finite ground theory. Then MinVEq-*ACUIG *and MinV-*ACUIG *are NP-hard w.r.t. term complexity.*

Proof. We restrict the attention to MinVEq-ACUIG (since MinV-ACUIG can be treated similarly), and reduce MinVEq-ACUI to MinVEq-ACUIG. Thus, let G be a fixed finite set of ground identities. Given an ACUI-unification problem Γ, we can assume without loss of generality that Γ contains none of the constants occurring in G (otherwise, we just rename the free constants in Γ). We now view Γ as ACUIG-unification problem.

To show correctness of the reduction, first assume that σ is a substitution that uses only constants occurring in Γ, and which violates at most ℓ equations of Γ. Equations $s \approx^? t \in \Gamma$ that are not violated satisfy $\sigma(s) \approx_{\mathsf{ACUI}} \sigma(t)$, and thus also $\sigma(s) \approx_{\mathsf{ACUIG}} \sigma(t)$. By Proposition 2.23, this implies that these equations are also not violated modulo G by σ. Thus, if we view Γ as ACUIG-unification problem, then σ violates at most ℓ equations of Γ modulo G.

Second, assume that σ is a substitution that uses only constants occurring in Γ and G, and which violates at most ℓ equations of Γ modulo G. Consider the substitution σ' that is obtained from σ by replacing every constant from G by $\mathbf{0}$. We claim that all equations not violated modulo G by σ are not violated by σ'. Thus, assume that $s \approx^? t \in \Gamma$ is an equation not violated by σ, which means that

$$S(\sigma(s))^G = S(\sigma(t))^G. \tag{2.3}$$

To show that this implies $S(\sigma'(s)) = S(\sigma'(t))$, assume that $c \in S(\sigma'(s))$. By our construction of σ' and the fact that theory constants from G do not occur in Γ, we know that c does not occur in G. In addition, $c \in S(\sigma'(s))$ implies $c \in S(\sigma(s))^G$, and thus (2.3) yields $c \in S(\sigma(t))^G$. Since saturation w.r.t. G can only add theory constants from G, we obtain $c \in S(\sigma(t))$. Thus, c is contained in s or introduced by σ. The latter implies that c is also introduced by σ' since only theory constants are removed when going from σ to σ'. Thus, we have shown that $S(\sigma'(s)) \subseteq S(\sigma'(t))$. Since the other inclusion can be shown analogously, this shows that $s \approx^? t$ is not violated by σ'. Consequently, σ' violates at most ℓ equations of Γ. □

Together with Proposition 2.33, this lemma yields the following complexity results.

Theorem 2.35. *The problems MinVEq-*ACUIG *and MinV-*ACUIG *are NP-complete both w.r.t. term complexity and w.r.t. combined complexity.*

2.5.2 The problem MinVEl-ACUIG

For the setting where the number of violating elements is minimized, the situation is less clear. Proposition 2.33 yields an NP upper bound for combined complexity. However, since MinVEl-ACUI is in P, a reduction from it would not yield a matching NP lower bound for MinVEl-ACUIG. One could try to adapt the polynomial-time algorithm for MinVEl-ACUI to MinVEl-ACUIG. Recall that this algorithm is based on considering projections of the equations in Γ to the free constants, and then solving the projected equations separately. While this would still work for the free constants in an ACUIG-unification problem, we now also need to consider the theory constants from G. For these, the separation into different equations does not work since the addition of constants by saturation could not be taken into account. This problem is illustrated in the next example.

Example 2.36. *Consider the system of equations*

$$\Gamma = \{a + x_2 \approx^? x_1,\ b + x_1 \approx^? c + f + g,\ c + x_2 \approx^? c + e\},$$

and the set of ground identities

$$G = \{a + b + c \approx d,\ b + c + e \approx f\}$$

from Example 2.26. As shown in that example, Γ *does not have an* ACUIG*-unifier. Thus, it makes sense to look for approximate unifiers.*

If we tried to solve the projected systems of equations (as described in Section 2.3.2) independently for each constant modulo ACUIG*, then we would see that* 5 *out of the* 7 *systems are not solvable: the systems for* a, b, e, f, g *are not solvable, whereas the ones for* c, d *are. This yields the substitution* $\theta = \{x_1 \mapsto c, x_2 \mapsto c\}$*, w.r.t. which 5 constants violate* Γ *(see the proof of Lemma 2.14), and it also happens that only these 5 constants violate* Γ *modulo* G*.*

However, by looking at the projections separately, we partially lose the possibility to prevent a violation by using G *to add the violating constant to the side where it does not occur syntactically. For this, other constants than the violating one may need to be present. Conversely, constants added by saturation may increase the number of violating constants, and their addition may depend on the presence of several constants.*

In our example, we can actually find a substitution w.r.t. which only 2 constants violate Γ*. If we set* $\sigma(x_1) := \sigma(x_2) := c + e$*, then the first equation is violated by* a*, and the second by* g*, while the third is not violated at all. Thus, w.r.t.* σ *there are only two violating constants. The reason is that* b, e, f *do not violate the second equation due to saturation w.r.t.* G*:* $\sigma(b + x_1) = b + c + e$ *and* $\sigma(c + f + g) = c + f + g$*, but saturation yields* $S(\sigma(b + x_1))^G = \{b, c, e, f\}$ *and* $S(\sigma(c + f + g))^G = \{b, c, e, f, g\}$*, and thus only* g *violates this equation modulo* G*.* ♢

As illustrated by this example, the approach used for MinVEl-ACUI to get a polynomial-time decision procedure in general does not work if we add ground identities. We will now show that, in fact, there is a fixed finite set of ground identities for which MinVEl-ACUIG is NP-hard w.r.t. term complexity. This obviously implies that the problem is also NP-hard for combined complexity. We will use a reduction from the NP-complete problem of 3-colorability [GJ90] to prove the hardness result.

Definition 2.37 (3-colorability).

Given: *a finite graph* $H = (X, E)$*, where* X *is the set of vertices and* $E \subseteq X \times X$ *is the set of edges of* H*.*

Question: *does there exist a* proper coloring *of* H *with three colors (say* a, b, c*), i.e., an assignment* $\sigma : X \to \{a, b, c\}$ *such that* $\sigma(x) \neq \sigma(y)$ *for every* $(x, y) \in E$*?* ♢

The fixed ground theory G_{3c} used to encode the "nature" of 3-colorability uses, as constants, the colors a, b, c and additionally d, e, f and k_{ij} with $i, j \in \{a, b, c\},\ i \neq j$. Let C be the set of these 12 constants. The theory G_{3c} then consists of the following identities:

- for every $i \in \{a, b, c\}$

$$i + d \approx f, \tag{2.4}$$

- for every $i, j \in \{a, b, c\}, i \neq j$

$$i + j + k_{ij} \approx f, \tag{2.5}$$
$$i + j + e \approx f. \tag{2.6}$$

We will explain the intuition underlying these identities once we have introduced the ACUIG_{3c}-unification problem. For the moment, just note that G_{3c} in fact does not depend on the input graph, i.e., it is fixed in the sense that the same theory G_{3c} is used for every input graph. The input graph $H = (X, E)$ is represented by the unification problem, where the vertices from X are used as variables.

To be more precise, given an the input graph $H = (X, E)$, the corresponding ACUIG_{3c}-unification problem Γ_H consists of the following equations:

- for every $x \in X$

$$x + d \approx^? f, \tag{2.7}$$

- for every $x \in X$ and $i, j \in \{a, b, c\}, i \neq j$

$$x + k_{ij} \approx^? \mathbf{0}, \tag{2.8}$$

- for every $(x, y) \in E$

$$x + y + e \approx^? f, \tag{2.9}$$

- finally, Γ_H also contains the equations

$$a + b + c \approx^? \mathbf{0}, \tag{2.10}$$
$$d + e \approx^? \mathbf{0}, \tag{2.11}$$
$$\sum_{i,j \in \{a,b,c\}, i \neq j} k_{ij} \approx^? \mathbf{0}. \tag{2.12}$$

At first sight, it may seem strange to have ground equations as part of a unification problem since they either hold modulo the given equational theory or are violated, independently of what substitution is used. However, in the context of MinVEl-ACUIG_{3c}, such equations can have a relevant effect. In fact, the equations (2.10)-(2.12) force all elements except for f to be violating. Intuitively, the question is then whether we can keep f from being violating. If this is the case, for every $x \in X$, the identities of G_{3c} together with equation (2.7) ensure that x is assigned at least one of a, b, c. Conversely, identities (2.5) together with equations (2.8) ensure that x is assigned at most one of a, b, c. Consequently, every variable (i.e., vertex) is assigned a unique color. Furthermore, identities (2.6) together with equation (2.9) ensure that x and y are assigned different colors, if $(x, y) \in E$.

Lemma 2.38. *The graph $H = (X, E)$ has a proper coloring iff there is a substitution σ such that at most 11 constants violate the ACUIG_{3c}-unification problem Γ_H modulo G_{3c} w.r.t. σ.*

Proof. First, note that there is a substitution σ such that at most 11 constants violate the ACUIG_{3c}-unification problem Γ_H modulo G_{3c} w.r.t. σ iff the constant f does not violate Γ_H modulo G_{3c}.

Now, assume that $H = (X, E)$ has a proper coloring, that is an assignment $\sigma : X \to \{a, b, c\}$ such that $\sigma(x) \neq \sigma(y)$ for every $(x, y) \in E$. Note that σ can also be viewed as a substitution. We claim that f does not violate Γ_H modulo G_{3c} w.r.t. this substitution. To see this, let us go through the equations of Γ_H:

- Equations of the form (2.7): for every $x \in X$, we have $\sigma(x) \in \{a, b, c\}$, and hence one of the identities (2.4) ensures that f is not violating the equation $x + d \approx^? f$ modulo G_{3c} w.r.t. σ.
- Equations of the form (2.8): such an equation $x + k_{i,j} \approx^? \mathbf{0}$ could only be violated by f modulo G_{3c} w.r.t. σ if an identity of the form (2.5) produced f on the left-hand side; this is not possible since $\sigma(x)$ contains only one of the constants from $\{a, b, c\}$ and not two different ones.
- Equations of the form (2.9): for every $(x, y) \in E$, since $\sigma(x), \sigma(y) \in \{a, b, c\}$ and $\sigma(x) \neq \sigma(y)$, one of the identities (2.6) ensures that f does not violate the equation $x + y + e \approx^? f$ modulo G_{3c} w.r.t. σ.
- f clearly does not violate equations (2.10)-(2.12) modulo G_{3c}.

Conversely, assume that $\sigma : X \to T_\Sigma(C)$ is a substitution such that f does not violate Γ_H modulo G_{3c} w.r.t. σ. Since, for every $x \in X$, f does not violate equation (2.8), we know that f does not appear in $\sigma(x)$. Furthermore, $\sigma(x)$ contains no more than one of a, b, c; otherwise, an identity of the form (2.5) would cause f to be violating. Moreover, since f does not violate equation (2.7) either, this implies that one of the identities (2.4) was activated, and hence $\sigma(x)$ contains exactly one of a, b, c. Overall, we can thus define the coloring τ by setting $\tau(x)$ to be the unique element of $\{a, b, c\}$ that appears in $\sigma(x)$. It remains to show that τ is a proper coloring of H: since for every $(x, y) \in E$, f does not violate equation (2.9), we know that one of the identities (2.6) was activated, and hence $\tau(x) \neq \tau(y)$. This completes the proof of the lemma. □

This lemma shows that MinVEl-ACUIG$_{3c}$ is NP-hard w.r.t. term complexity. Together with Proposition 2.33, we thus obtain the following theorem.

Theorem 2.39. *There exists a ground theory G_{3c} for which MinVEl-*ACUIG$_{3c}$ *is NP-complete w.r.t. term complexity. This implies that MinVEl-*ACUIG *is NP-complete w.r.t. combined complexity.*

Note, however, that there are finite ground theories G for which MinVEl-ACUIG is in P w.r.t. term complexity. In fact, our results in Section 2.3.2 show that the empty ground theory $G = \emptyset$ is such a theory.

Similarly, if we consider a theory G containing only identities of the form $a \approx b$ for $a, b \in C$ for a finite set of constants C, then MinVEl-ACUIG is in P w.r.t. term complexity. Basically, the idea is to reduce MinVEl-ACUIG to MinVEl-ACUI by replacing all occurrences of equivalent constants by a single representative of the class. In the resulting system of equations there is no interaction between the different constants occurring in it, and hence it can be treated as a MinVEl-ACUI problem. An optimal substitution for this instance is also optimal for the original one, although the actual number of violating elements modulo G may be higher in the original problem. In fact, if b is the chosen representative of an equivalence class of

constants, and b violates an equation in the MinVEl-ACUI instance, than all then elements of the class violate this equation modulo G.

In order to provide less trivial examples of ground theories G for which MinVEl-ACUIG can be decided in polynomial time, we restrict the syntactic form of the ground identities that may occur in G.

Definition 2.40. *The finite set of ground identities is called* unary *if every identity in G is of the form $a \approx a + t$ for a constant a and a ground term t.* ♢

Note that unary theories actually also cover the case of identities of the form $a \approx b$. In fact, it is easy to see that the theory $G_i = \{a \approx b\}$ is equivalent to the unary theory $G_u = \{a \approx a + b, b \approx b + a\}$ in the presence of ACUI:

- we have $a \approx_{\mathsf{ACUIG}_u} a + b \approx_{\mathsf{ACUIG}_u} b + a \approx_{\mathsf{ACUIG}_u} b$,
- and $a \approx_{\mathsf{ACUIG}_i} a + a \approx_{\mathsf{ACUIG}_i} a + b$ as well as $b \approx_{\mathsf{ACUIG}_i} b + b \approx_{\mathsf{ACUIG}_i} b + a$.

The name "unary" for theories satisfying the above definition stems from the fact that, for such theories, the saturation rules are unary in the sense that their applicability depends on the presence of only a single constant. Indeed, an identity of the form $a \approx a + t$ induces a rule that is applicable whenever a occurs in a given set of constants, and its effect is to add the constants occurring in t to this set (unless they are already there). Since a occurs also on the right-hand side, there is no saturation rule that is triggered by the right-hand side of such an identity.

Lemma 2.41. *Let G be a unary ground theory and $s, t \in T_\Sigma(F)$. Then $S(s+t)^G = S(s)^G \cup S(t)^G$. In particular, this implies that $S(s)^G = \bigcup_{a \in S(s)} \{a\}^G$.*

Proof. The inclusion from right to left follows from Lemma 2.22, and thus holds in general (i.e., also for non-unary theories): $S(s)^G \cup S(t)^G \subseteq (S(s) \cup S(t))^G = S(s+t)^G$.

To show the other direction, assume that $a \in S(s+t)^G$. We show $a \in S(s)^G \cup S(t)^G$ by induction on the number of saturation steps needed to add a to $S(s+t)^G$. In the base case, we have $a \in S(s+t) = S(s) \cup S(t) \subseteq S(s)^G \cup S(t)^G$. Thus, assume that a is added in step n of the saturation process, and that all the constants that have been added previously are contained in $S(s)^G \cup S(t)^G$. Thus, there is a constant $b \in S(s)^G \cup S(t)^G$ and an identity $b \approx b + u \in G$ with $a \in S(u)$. If $b \in S(s)^G$, then the existence of the identity $b \approx b + u$ in G and the fact that $S(s)^G$ is saturated imply that $a \in S(s)^G$. In the same way, $b \in S(t)^G$ implies that $a \in S(t)^G$. □

Note that this lemma need not hold for non-unary ground theories. An easy counterexample is $G = \{a + b \approx c\}$, where $c \in S(a+b)$, but $c \notin S(a) \cup S(b)$.

Given a unary ground theory G, we will show that MinVEl-ACUIG can be reduced to MinVEl-ACUI. Before we can describe this reduction, we need to extend the notion of saturation from sets of constants to terms, substitutions, and unification problems. Given a ground term $s \in T_\Sigma(F)$, its saturation is $s^G := \sum(S(s)^G)$. Note that we have $S(s^G) = S(s)^G$ for any $s \in T_\Sigma(F)$, and that Lemma 2.41 yields $(s_1 + \ldots + s_n)^G \approx_{\mathsf{ACUIG}} s_1^G + \ldots + s_n^G$ for $s_1, \ldots, s_n \in T_\Sigma(F)$.

For a ground substitution $\sigma : X \to T_\Sigma(F)$ we define its saturation σ^G as $\sigma^G(x) := \sigma(x)^G$ for every $x \in X$. Recall that a term $t \in T_\Sigma(F, V)$ can be written in the form $t = t_0 + x_1 + \ldots + x_n$,

where $t_0 \in T_\Sigma(F)$ and $x_1, \ldots, x_n \in V$. We set $t^G := t_0^G + x_1 + \cdots + x_n$. Finally, given an ACUIG-unification problem with constants Γ, its saturation Γ^G consists of the equations $s^G \approx^? t^G$ for every $s \approx^? t$ in Γ.

The following lemma is an easy consequence of Lemma 2.41.

Lemma 2.42. *Let G be a unary ground theory, $t \in T_\Sigma(F)$, and σ a ground substitution. Then $S(\sigma^G(t^G)) = S(\sigma(t))^G$.*

Proof. Let t be of the form $t = t_0 + x_1 + \cdots + x_n$ for a ground term t_0. Then we have

$$\begin{aligned} S(\sigma^G(t^G)) &= S(t_0^G + \sigma^G(x_1) + \cdots + \sigma^G(x_n)) \\ &= S(t_0^G + \sigma(x_1)^G + \cdots + \sigma(x_n)^G) = S(\sigma(t)^G) = S(\sigma(t))^G. \end{aligned}$$ □

The idea is now to reduce MinVEl-ACUIG for Γ to MinVEl-ACUI for Γ^G. The following theorem states correctness of this reduction.

Theorem 2.43. *Let G be a unary ground theory, Γ an* ACUIG*-unification problem, and $\ell \geq 0$. Then the following are equivalent:*

1. *There is a substitution σ such that at most ℓ constants violate Γ modulo G w.r.t. σ.*
2. *There is a substitution θ such that at most ℓ constants violate Γ^G w.r.t. θ.*

Proof. Assume initially that there exists a substitution σ w.r.t. which Γ has at most ℓ violating elements modulo G. It is easy to see that σ^G is a substitution w.r.t. which Γ^G has at most ℓ violating elements. In fact, Lemma 2.42 yields for every $s \approx^? t \in \Gamma$

$$S(\sigma^G(s^G)) \,\Delta\, S(\sigma^G(t^G)) = S(\sigma(s))^G \,\Delta\, S(\sigma(t))^G,$$

and hence Γ^G (viewed as a MinVEl-ACUI instance) has at most ℓ violating constants w.r.t. σ^G.

For the opposite direction, assume that there exists a substitution θ w.r.t. which Γ^G has at most ℓ violating elements (not considering G). Recall that, in Lemma 2.14.3, when constructing an optimal substitution for a MinVEl-ACUI problem, we only use non-violating elements. Hence, we can assume without loss of generality that θ introduces no violating constants. Under this assumption, we will now show that, also modulo G, Γ has at most ℓ violating elements w.r.t. θ. Indeed, assume that a does not violate Γ^G w.r.t. θ. Then, for every $s \approx^? t \in \Gamma$ we have that $a \notin S(\theta(s^G)) \,\Delta\, S(\theta(t^G))$.

- If $a \in S(\theta(s^G)) \cap S(\theta(t^G))$, we obtain by Lemma 2.42 that $a \in S(\theta(s))^G \cap S(\theta(t))^G$, and hence a does not violate $s \approx^? t$ modulo G.
- If $a \notin S(\theta(s^G)) \cup S(\theta(t^G))$, it is still possible that $a \in S(\theta(x)^G)$ for some x occurring in s (the case where x occurs in t is treated similarly), and hence $a \in S(\theta(s))^G$. By Lemma 2.41, we obtain that there exists some $b \in S(\theta(x))$ such that $a \in \{b\}^G$. Since we assume without loss of generality that θ does not introduce any violating elements, b is also not violating. Hence, b occurs in $\theta(t^G)$, and thus also in $\theta(t)^G$, which again by Lemma 2.41 implies that $a \in S(\theta(t))^G$.

  If, on the other hand, $a \notin S(\theta(x))^G$ for all variables x occurring in s or t, then we have $a \notin S(\theta^G(s^G)) \cup S(\theta^G(t^G))$, which by Lemma 2.42 is the same as $a \notin S(\theta(s))^G \cup S(\theta(t))^G$.

In any case, we conclude that $a \notin S(\theta(s))^G \,\Delta\, S(\theta(t))^G$, and hence a does not violate $s \approx^? t$ modulo G. To sum up, we have shown that every non-violating element of Γ^G w.r.t. θ remains non-violating modulo G for Γ w.r.t. θ, and thus the upper bound of ℓ for violating elements also holds in Γ modulo G. □

Since saturation can be done in polynomial time (even if G is seen as part of the input) and MinVEl-ACUI can be solved in polynomial time, this theorem yields the following complexity result for the class of unary ground theories.

Corollary 2.44. *Restricted to unary ground theories G, MinVEl-*ACUIG *is in P w.r.t. combined complexity, and thus also w.r.t. term complexity.*

2.6 Outlook

In this chapter, we have showcased our program in the setting of ACUI in order to obtain some initial results and some intuition before we venture into the more expressive setting of $\mathcal{FL}_0$. We have extended ACUI-unification in two directions. On the one hand, we have considered approximate ACUI-unification w.r.t. three different ways of measuring the degree to which the equations of the unification problem are violated by a given substitution that is not a unifier. For two of these measures, the complexity of the associated decision problem increases from P to NP-complete, whereas for one of them it stays in P. In essence, approximate unification with respect to this measure reduces to investigating the different constants separately. This is in line with approximate unification in $\mathcal{FL}_0$, as we will see in Chapter 6, where one can reduce the initial problem to independently solving language equations, one for every concept name, hence maintaining the same complexity as for the classical case.

On the other hand, we have extended ACUI-unification to ACUIG-unification, i.e., unification in equational theories that are obtained from ACUI by adding a finite set G of ground identities. We were able to show that adding such identities does not change the complexity of the unification problem. Finally, we have combined the two extensions, i.e., we have investigated approximate ACUIG-unification. For the measures for which already approximate ACUI-unification is NP-complete, the same holds for approximate ACUIG-unification. For the third measure, the situation turns out to be more interesting. We were able to show that there is a finite set G of ground identities such that approximate ACUIG-unification for this fixed theory is NP-complete. But we have also introduced a class of ground theories G for which approximate ACUIG-unification is in P.

In the setting of this third measure, it would be interesting to see whether one can show a dichotomy result, i.e., whether one can prove that, depending on which ground theory G is used, the complexity of approximate ACUIG-unification is either in P or NP-complete. If this is actually the case, the next step would be to attempt a classification of the two cases, i.e., come up with conditions that ensure membership in P or NP-completeness.

Chapter 3

Languages and Automata

We now want to leave the safety of ACUI and venture into the world of $\mathcal{FL}_0$. As discussed in the Introduction, in this setting, terms no longer correspond to subsets of a main set of elements, but rather to (tuples of) languages, i.e. sets of words over some alphabet. Before we introduce the actual correspondence between concepts and languages, we will provide some technical results on how to deal with languages.

Even though $\mathcal{FL}_0$ concepts correspond to finite languages in the classical (no TBox) setting, we have noted already that TBoxes may cause these languages to become infinite. Furthermore, our approach for approximate unification in Chapter 6 eventually reduces to checking for the existence of arbitrary (i.e., not necessarily finite) approximate solutions to language equations. Finally, our algorithm for deciding matching in the presence of TBoxes in Chapter 8 is taking a detour through the DL $\mathcal{FL}_{\text{reg}}$, in which concept descriptions correspond to (tuples of) infinite languages. In order to define concept distance measures with respect to general TBoxes in Chapter 5 we will need to compute distances between such infinite languages. Apart from the motivation from Description Logic, the results we will present can be of independent interest from a language-theoretic point of view.

In this chapter, after we provide some basic definitions from Formal Language Theory and some elementary results regarding finite automata, we will introduce the notion of language distances, by further investigating three particular such distances.

If we have access to a procedure that actually computes the distance between languages, we first need to be able to specify these languages in a finite way before we feed them as input to the procedure. Hence, we will then describe a correspondence between (tuples of) languages and infinite trees, and how tree automata can be utilized to finitely represent them. Afterwards, we will introduce weighted tree automata, which provide a mechanism for assigning values to infinite trees, and demonstrate how they can be used to define language distances. To the best of our knowledge, there is no technique for computing the behavior of such automata; this will be the final focus of this chapter.

3.1 Basic Definitions

Let Σ be a finite nonempty set that we will call *alphabet* and its elements *letters*. A *word* over an alphabet Σ is a finite sequence consisting of zero or more letters of Σ, where the same letter might occur several times. The sequence of zero letters is called the *empty word* and is denoted by ε. For example, ε, 1, 00, 10110 are words over the binary alphabet $\{0, 1\}$. The set of all words over an alphabet Σ is denoted by Σ^*. If w and v are words over Σ, then so is their *concatenation* wv, obtained by juxtaposition , that is, writing w and v one after another. Concatenation is an associative operation and the empty word ε acts as an identity:

$w\varepsilon = \varepsilon w = w$ holds for all words w. Since associativity holds, powers of the form w^n are defined in the usual way: by definition, $w^0 := \varepsilon$, and $w^{k+1} := w^k w$.

The *length* of a word w, denoted by $|w|$, is the number of letters in w when each letter is counted as many times as it occurs. Note that in particular $|\varepsilon| = 0$.

A *(formal) language* over Σ is a (finite or infinite) subset of Σ^*. As usual in mathematics, the set of all languages, i.e., the powerset of Σ^*, is denoted by 2^{Σ^*}. For example, $\emptyset$, $\{\varepsilon\}$, $\{1, 00, 10110\}$, $\{1^i 0 \mid i \text{ is even}\}$, $\{10^i 1 \mid i \text{ is prime}\}$ are languages over the binary alphabet $\{0, 1\}$. Note that the first three languages are finte, while the last two are infinite. A finite language can always be defined by listing all its words. Such a procedure is not possible for infinite languages. Some finitary specification (other that simple listing) is required to define an infinite language. Much of formal language theory deals with such finitary specifications of infinite languages: automata, grammars, etc. In this thesis, we deal with languages that are representable by automata.

Various *operations* are defined for languages. Regarding languages as sets, the *Boolean* operations of *union, intersection,* and *complement* (with respect to Σ^*), and furthermore those of *difference* and *symmetric difference* are immediately defined in the usual fashion: Given languages K, L we have

$$K \cup L = \{a \in \Sigma^* \mid a \in K \text{ or } a \in L\} \qquad K \setminus L = \{a \in K \mid a \notin L\}$$
$$K \cap L = \{a \in \Sigma^* \mid a \in K \text{ and } a \in L\} \qquad K \Delta L = (K \setminus L) \cup (L \setminus K)$$
$$K^c = \{a \in \Sigma^* \mid a \notin K\}$$

The operation of *concatenation* is extended from words to languages in the natural way:

$$L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}.$$

As usual in formal language theory, we will often omit the "$\cdot$" operator, and simply write $L_1 L_2$ instead of $L_1 \cdot L_2$. As in the word setting, this operation is associative, and the language $\{\varepsilon\}$ acts as an identity: $L\{\varepsilon\} = \{\varepsilon\}L = L$ holds for all languages L. Powers of the form L^n of languages are similarly extended. Note that $L^0 = \{\varepsilon\}$. The *Kleene star* of a language L, in symbols L^* is defined to be the union of all non-negative powers of L, i.e,

$$L^* = \bigcup_{i=0}^{\infty} L^i.$$

Observe that this definition is in accordance with the notation used earlier: if we view Σ as the finite language whose words are singleton letters, then Σ^* is the set of words built by concatenating arbitrarily many "words" (letters) from Σ, i.e., all words over Σ.

Furthermore, we will denote the set of all words over Σ of length at least (respectively, at most/less than/more than) m by $\Sigma^{\geq m}$ (respectively, $\Sigma^{\leq m}/\Sigma^{<m}/\Sigma^{>m}$).

Another operation that can be defined for languages is that or *mirroring*. For a word $w = \sigma_1 \dots \sigma_\ell$, its mirror image is defined as $w^{mi} = \sigma_\ell \dots \sigma_1$, and for a language L its mirror image is $L^{mi} = \{w^{mi} \mid w \in L\}$.

Finally, the *left-quotient* of a language L over Σ with a word $w \in \Sigma^*$ is defined to be the set $w^{-1}L = \{v \in \Sigma^* \mid wv \in L\}$.

3.2 Finitely representing languages: finite automata and regular expressions

As discussed earlier, one of the main focuses of formal language theory is how to specify an infinite language in a finite way. Of course, it is not possible to derive such a specification for every possible language[1]. In this section we restrict our focus to *regular languages*. There are two major types of mechanisms for defining languages: acceptors and generators. Next, we introduce *finite automata*, simple devices which accept regular languages, and *regular expressions*, which generate regular languages, and provide some basic results on how to transit from one to the other. The definitions and the results in this section can be found in any handbook on formal languages or even theoretical computer science [Per90; Yu97; HMU06].

Finite automata A finite automaton consists of a finite set of internal states and a set of rules that govern the change of the current state when reading a given input symbol. If the next state is always uniquely determined by the current state and the current input symbol, we say that the automaton is deterministic; otherwise it is called nondeterministic. Depending on the state the automaton ends up after reading the entire input word, it either accepts or rejects the word.

More formally, we have the following:

Definition 3.1. *A* deterministic finite automaton (DFA) *A is a* 5*-tuple $(Q, \Sigma, \delta, q_0, F)$, where*

- *Q is a finite set of states,*
- *Σ is the finite input alphabet,*
- *$\delta : Q \times \Sigma \to Q$ is the state transition function,*
- *$q_0 \in Q$ is the starting state, and*
- *$F \subseteq Q$ is the set of final states.* ♢

Note that δ describes the functionality of the automaton, that is, how the internal states change according to the symbol from Σ that is currently being read. Intuitively, a DFA accepts a word if, starting from the initial state, after reading the input word letter by letter, it ends up in a final state. More formally, we can extend δ to a function operating on words $\delta^* : Q \times \Sigma^* \to Q$ by setting

- $\delta^*(q, \varepsilon) = q$ for every $q \in Q$,
- $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ for every $q \in Q$, $w \in \Sigma^*$ and $a \in \Sigma$.

We say that a word w is *accepted* by the DFA A, if $\delta^*(q_0, w) \in F$. The *behavior* of A, in symbols $\mathcal{L}(A)$, is the language consisting of all the words that A accepts, i.e., $\mathcal{L}(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$. Conversely, we say that a language L is *recognizable* if there exists a DFA A such that $L = \mathcal{L}(A)$. In this case, we say that L is *recognized* or *accepted* by A.

[1] See *undecidable languages* [Pap94].

Nondeterministic finite automata *NFAs* generalize DFAs by allowing several initial states, and multiple possible transitions for a given state and input symbol. Formally, a *nondeterministic finite automaton (NFA)* is a 5-tuple $M = (Q, \Sigma, \Delta, I, F)$, where Q, Σ, and F are defined the same way as for a DFA, $I \subseteq Q$ is a *set* of initial states, and $\Delta : Q \times \Sigma \to 2^Q$ is the transition function, where 2^Q denotes the powerset of Q. Intuitively, when the automaton is in state q and reads input letter a, then it can choose to transit to any of the states in $\Delta(q, a)$.

Overall, an NFA (potentially) allows for multiple ways to process an input word. It may start with a different initial state, and at every input symbol it might have many possible succeeding states. Hence, it is possible that whether the automaton ends up in a final state will depend on the choices that were made. By definition, an NFA accepts a word if it is possible starting from some initial state to reach a final one after reading the input word.

Formally, similarly to DFAs, we can extend Δ into a function Δ^* operating on words, in order to define the behavior of the NFA. In particular,

- $\Delta^*(q, \varepsilon) = \{q\}$ for every $q \in Q$,
- $\Delta^*(q, wa) = \{q' \in \Delta(p, a) \mid p \in \Delta^*(q, w)\}$ for every $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.

A word w is *accepted* by the NFA A, if $\Delta^*(q_0, w) \cap F \neq \emptyset$ for some $q_0 \in I$, and the behavior of an NFA is defined similarly.

Every DFA can be viewed as an NFA, where each value of the transition function is a singleton.

Two automata are said to be *equivalent* if they accept exactly the same language.

Lemma 3.2 (well-known). *For each NFA there exists an equivalent DFA of at most exponential size. Furthermore, there are NFAs for which such a blowup is unavoidable.*

Regular expressions Regular expressions are succinct and comprehensible expressions in sequential form that specify languages. They were first introduced by Kleene [Kle56].

Regular expressions are built from the letters of the alphabet by using the operations of union, concatenation and Kleene star. More formally:

- The empty set $\emptyset$ is a regular expression denoting the empty language.
- Every letter $a \in \Sigma$ is a regular expression denoting the singleton language $\{a\}$.
- Let e_1, e_2, e be regular expressions denoting the languages $L(e_1), L(e_2), L(e)$ respectively. Then, $e_1 \cup e_2$, $e_1 e_2$ and e^* are also regular expressions that denote the languages $L(e_1) \cup L(e_2)$, $L(e_1)L(e_2)$ and $L(e)^*$ respectively.

Note that all finite languages can be expressed by a regular expression. In particular, $\{\varepsilon\} = \emptyset^*$.

Equivalence and complexity issues In [Kle56], Kleene has shown that the family of languages specified by regular expressions and recognizable languages coincide, a class that is refered to as simply *regular* languages. The proof of this result, which is known as *Kleene's theorem*, involves constructing regular expressions that correspond to the behavior of a given automaton and vice versa.

Kleene mentions that, even though he does not investigate how big the resulting expressions and automata have to be, this is an interesting topic to study. Indeed, a lot of research has been devoted in this area since then. Next, we provide some basic results on the complexity of such constructions.

In order to provide formal statements we need a firm definition of size for the notions involved. The size of a regular expression is the amount of symbols it contains, while the size of an automaton (either DFA or NFA) is usually considered to be the cardinality of its set of states.[2]

Lemma 3.3. (Equivalence between finite automata and regular expressions)

- *Given a DFA/NFA A, one can construct a regular expression e_A of size exponential in the size of A such that $L(e_A) = \mathcal{L}(A)$.*
- *Given a regular expression e, one can construct an NFA A_e of size polynomial in the size of e such that $\mathcal{L}(A_e) = L(e)$. By Lemma 3.2, this implies that a DFA of exponential size can be constructed with the same property.*

For a given regular language there are several automata recognizing it. However, among equivalent DFAs there always exists one has minimal amount of states.

Lemma 3.4 (well-known). *For each regular language, there exists a minimal DFA that accepts it, that is, a DFA with a minimum number of states and this DFA is unique (up to renaming of the states). Given a DFA, the minimal equivalent DFA can be obtained in polynomial time.*

Furthermore, of particular interest is the complexity of certain language operations, i.e., given automata accepting certain languages, how large does an automaton have to be in order to accept the result of the operation of the languages. The following results are well-known in automata theory, and most can be found in [YZS94] and [HK03].

Lemma 3.5. (State complexity of language operations)

- *Given DFAs A and B with m and n states respectively, one can construct a DFA C with mn states such that $\mathcal{L}(C) = \mathcal{L}(A) \cup \mathcal{L}(B)$, and also for $\mathcal{L}(C) = \mathcal{L}(A) \cap \mathcal{L}(B)$.*
- *Given DFAs A and B with m and n states respectively, one can construct a DFA C with $m2^n - 2^{n-1}$ states such that $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$.*
- *Given DFA A with n states, one can construct a DFA B with $2^{n-1} + 2^{n-2}$ states such that $\mathcal{L}(B) = \mathcal{L}(A)^*$.*
- *Given DFA A with n states, one can construct a DFA B with n states such that $\mathcal{L}(B) = \mathcal{L}(A)^c$.*
- *Given DFA A with n states, one can construct a DFA B with 2^n states such that $\mathcal{L}(B) = \mathcal{L}(A)^{mi}$.*
- *Given DFA A with n states and word $w \in \Sigma^*$, one can construct a DFA B with n states such that $\mathcal{L}(B) = w^{-1}\mathcal{L}(A)$. Note in particular that the size of w does not affect the size of the automaton.*

[2] There have been other definitions of the size of an automaton, which also include the size of the transition function, and/or the initial/final states. In any case, even if the size is defined this way, it can only be polynomially bigger than the size we define here, hence for our analysis our definition suffices.

- *Given NFAs A and B with m and n states respectively, one can construct an NFA C with* $m+n$ *states such that* $\mathcal{L}(C) = \mathcal{L}(A) \cup \mathcal{L}(B)$.
- *Given NFAs A and B with m and n states respectively, one can construct an NFA C with* mn *states such that* $\mathcal{L}(C) = \mathcal{L}(A) \cap \mathcal{L}(B)$.
- *Given NFAs A and B with m and n states respectively, one can construct an NFA C with* $m+n$ *states such that* $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$.
- *Given NFA A with n states, one can construct an NFA B with* $n+1$ *states such that* $\mathcal{L}(B) = \mathcal{L}(A)^*$.
- *Given NFA A with n states, one can construct an NFA B with* 2^n *states such that* $\mathcal{L}(B) = \mathcal{L}(A)^c$.
- *Given NFA A with n states, one can construct an NFA B with* $n+1$ *states such that* $\mathcal{L}(B) = \mathcal{L}(A)^{mi}$.
- *Given NFA A with n states and word* $w \in \Sigma^*$*, one can construct an NFA B with n states such that* $\mathcal{L}(B) = w^{-1}\mathcal{L}(A)$*. Note in particular that the size of w does not affect the size of the automaton.*
- *Given DFAs* $A_1, \ldots, A_k$ *each with at most n states, one can construct a DFA B with* n^k *states such that* $\mathcal{L}(B) = \bigcup_{i=1}^{k} \mathcal{L}(A_i)$*, and also for* $\mathcal{L}(B) = \bigcap_{i=1}^{k} \mathcal{L}(A_i)$.
- *Given NFAs* $A_1, \ldots, A_k$ *each with at most n states, one can construct an NFA B with* nk *states such that* $\mathcal{L}(B) = \bigcup_{i=1}^{k} \mathcal{L}(A_i)$.
- *Given NFAs* $A_1, \ldots, A_k$ *each with at most n states, one can construct an NFA B with* n^k *states such that* $\mathcal{L}(B) = \bigcap_{i=1}^{k} \mathcal{L}(A_i)$.
- *Given NFAs* $A_1, \ldots, A_k$ *each with at most n states, one can construct an NFA B with* nk *states such that* $\mathcal{L}(B) = \mathcal{L}(A_i) \cdots \mathcal{L}(A_k)$.

3.3 Language Distances

In this section we will discuss about functions that measure the distance between languages. Even though there has been quite some research on distance measures for *words*, mainly because of their applications in information retrieval and computational biology (see [Nav01] for a survey on the topic), the respective area for languages has received limited investigation.

A survey containing many ideas and examples of how such a measure can be defined is contained in [KN06]. In particular, the authors study measures based on the symmetric difference and on the Kolmogorov complexity of the input languages, as well as functions based on neighbourhood criteria and distances on words. Kephart in his PhD thesis [Kep05] also considers a measure that is based on topological entropy.

For the purposes of this thesis, we restrict our focus in the first case, i.e., functions that assess the size of the symmetric difference of languages.

Traditionally in mathematics, distance has been linked to the notion of a *metric*. Let S be a set. A function $d : S \times S \to \mathbb{R}_{\geq 0}$ is called a *metric* on S if it satisfies the following conditions for every $a, b, c \in S$:

$(M1)$ $d(a,b) = 0 \iff a = b$

$(M2)$ $d(a,b) = d(b,a)$

$(M3)$ $d(a,c) \leq d(a,b) + d(b,c)$

If the distance between two elements of S is not required to be finite, then a function d satisfying conditions $(M1)$-$(M3)$ is called a *generalized metric*.

Following the aforementioned tradition, we use the term *language distance* to denote a metric on the space of languages, i.e., a function $d : 2^{\Sigma^*} \times 2^{\Sigma^*} \to \mathbb{R}_{\geq 0}$ that takes as input two languages and outputs a positive real number.

If the underlying function is a generalized metric, we will talk about a *generalized language distance*.

A common approach to define a language distance function is to "measure" the size of the symmetric difference of the input languages [Via77; Kep05], i.e., define $d(K,L) := f(K \Delta L)$ where $K \Delta L := (K \setminus L) \cup (L \setminus K)$ and f is an appropriate function. This way, $(M2)$ is guaranteed to hold. If f is actually a measure (in the mathematical sense [Rud87]), $(M3)$ holds as well, (since $K \Delta M \subseteq K \Delta L \cup L \Delta M$ and f is a subadditive function). $(M1)$ corresponds to the requirement that $f(L) = 0$ iff $L = \emptyset$. Below we present some basic language distances based on this principle.

Arguably, the easiest way to "count" how far apart two languages are is to consider the cardinality of their symmetric difference. This is formalized by the function

$$d_0(K,L) = |K \Delta L|.$$

This approach, however, has two major disadvantages. On the one hand, when infinite languages are considered the value might become infinite. This is hence not a language distance, but rather a generalized one. On the other hand, this completely disregards the structure of the objects under consideration, that is the languages, and simply views them as sets.

Another function that also applies on domains whose elements are sets, but has received quite some interest in the literature [KN06], is the Jaccard distance [Jac12] defined as

$$d_J(K,L) = \frac{|K \Delta L|}{|K \cup L|}.$$

Note however that this function is only defined for finite sets, hence does not apply to languages in general.

The first approach that was tailored towards languages was published by Vianu [Via77], who applied a metric proposed earlier by Bodnarchŭk [Bod65]. This is in fact the most well-investigated language distance, since it appears in virtually every paper that investigates the area [Kep05; KN06; FK15]. The formal definition of this distance is

$$d_1(K,L) = 2^{-\ell},$$

where $\ell = \min\{|w| \mid w \in K \Delta L\}$ [3]. This function considers the length ℓ of the shortest word in the symmetric difference of K and L and yields $2^{-\ell}$ as distance, which becomes smaller if ℓ gets larger. The main intuition, which aligns well with our motivation from description

[3] As usual, we assume that $\min \emptyset = \infty$ and $2^{-\infty} = 0$.

logics[4], is that longer words are of less importance than shorter ones. One drawback of this function, however, is that d_1 only focuses on the length of the shortest difference, completely overlooking if there are more differences of equal or larger length.

This issue is addressed by the function

$$d_2(K,L) = \mu(K \,\Delta\, L)$$

where for $M \in 2^{\Sigma^*}$ we define

$$\mu(M) = \frac{1}{2} \sum_{w \in M} (2|\Sigma|)^{-|w|}.$$

This function takes into account all the elements of the symmetric difference, each adding some amount to the overall distance, but longer words count less that shorter ones. More precisely, the weight of the word u counts as much as the sum of the weights of all words uv properly extending u.

A very similar language distance has been considered in [KN06]. The authors of that work point out the dependency on the size of the total alphabet Σ as a potential drawback. As a remedy, adapting the Jaccard distance they suggest the function

$$\delta_2(K,L) = \sum_{n \in \mathbb{N}} \frac{\left|(K \,\Delta\, L)^{[n]}\right|}{\left|(K \cup L)^{[n]}\right|},$$

where for a language $L \subseteq \Sigma^*$ and a natural number $n \in \mathbb{N}$ we set $L^{[n]}$ to denote the projection of L on words of length n, i.e., $L^{[n]} = L \cap \Sigma^n = \{w \in L \mid |w| = n\}$. Using the projection notation, with simple calculations we can rewrite the measure used for d_2 as follows.

$$\begin{aligned} \mu(L) &= \frac{1}{2} \sum_{w \in L} (2|\Sigma|)^{-|w|} = \frac{1}{2} \sum_{n \in \mathbb{N}} \sum_{w \in L^{[n]}} \frac{1}{(2|\Sigma|)^{|w|}} \\ &= \frac{1}{2} \sum_{n \in \mathbb{N}} \sum_{w \in L^{[n]}} \frac{1}{(2|\Sigma|)^{n}} = \frac{1}{2} \sum_{n \in \mathbb{N}} \frac{\left|L^{[n]}\right|}{(2|\Sigma|)^{n}} = \frac{1}{2} \sum_{n \in \mathbb{N}} \frac{\left|L^{[n]}\right|}{2^n |\Sigma^n|} \end{aligned} \tag{$\star$}$$

More functions that are based on the symmetric difference of the input languages were considered in [KN06] and [Kep05; FK15], but they involve limit computations. In particular, this implies that they do not satisfy $(M1)$, and hence they do not constitute language distances under our definition.

For the rest of our analysis we will deal with d_0, d_1, and d_2. Next we show that these functions are indeed language distances.

Lemma 3.6. *The function d_0 is a generalized metric, and the functions d_1, d_2 are metrics.*

Proof. Initially note that all these functions are well-defined for any pair of languages. The value of d_0 is a nonnegative natural number or ∞, while d_1 and d_2 are bounded from above by 1. For d_1 this is easy to see since the length of the shortest word in the symmetric difference of two languages is a nonnegative natural number. For d_2, if we use the projected

[4] See Section 5.3

form of μ we derived in $(\star)$, since for every language L and $n \in \mathbb{N}$ we have that $L^{[n]} \subseteq \Sigma^n$, we obtain:

$$\mu(L) = \frac{1}{2}\sum_{n\in\mathbb{N}} \frac{\left|L^{[n]}\right|}{2^n\,|\Sigma^n|} \leq \frac{1}{2}\sum_{n\in\mathbb{N}} \frac{|\Sigma^n|}{2^n\,|\Sigma^n|} = \frac{1}{2}\sum_{n\in\mathbb{N}} \frac{1}{2^n} = \frac{1}{2}\cdot 2 = 1.$$

Next, in order to show that these functions are language distances, we have to show that they satisfy $(M1)$, $(M2)$ and $(M3)$. $(M1)$ and $(M2)$ are obvious. Regarding the triangle inequality $(M3)$ $d(K,L) \leqslant d(K,M) + d(M,L)$:
Initially note that for languages K, L, M it holds that $K \triangle L \subseteq K \triangle M \cup M \triangle L$. Indeed, if $x \in K \setminus L$ then either $x \notin M$, implying $x \in K \triangle M$, or $x \in M$, implying $x \in M \triangle L$.

For d_0 we immediately obtain

$$d_0(K,L) = |K \,\Delta\, L| \leq |K \triangle M \cup M \triangle L| \leq |K \triangle M| + |M \triangle L| = d_0(K,M) + d_0(M,L).$$

For d_1, if $d_1(K,L) = 0$ then $(M3)$ holds trivially. Suppose that $d_1(K,L) = 2^{-n}$, where $n = |w|$ for some $w \in K \triangle L$. Without loss of generality, suppose that $w \in K \triangle M$. Thus $n \geqslant \min\{\,|w| : w \in K \triangle M\,\}$ and consequently $d_1(K,M) \geqslant 2^{-n}$. Consequently, $d_1(K,L) \leqslant d_1(K,M) + d_1(M,L)$.

For d_2, since $K \triangle L \subseteq K \triangle M \cup M \triangle L$, summing over these sets, we get that

$$\sum_{w\in K\triangle L} (2|\Sigma|)^{-|w|} \leqslant \sum_{w\in K\triangle M \cup M\triangle L} (2|\Sigma|)^{-|w|} \leqslant \sum_{w\in K\triangle M} (2|\Sigma|)^{-|w|} + \sum_{w\in M\triangle L} (2|\Sigma|)^{-|w|}$$

and thus $d_2(K,L) \leqslant d_2(K,M) + d_2(M,L)$. □

An important property for language distances, which will come in handy in later chapters, is monotonicity w.r.t. the symmetric difference of the input languages. Formally, the property mentions that

$$K \,\Delta\, L \subseteq M \,\Delta\, N \implies d(K,L) \leq d(M,N). \tag{3.1}$$

Note that this property holds for any language distance function d that is of the form $d(K,L) = m(K \,\Delta\, L)$ where m is a measure, or at least a monotone function. In particular, this implies that d_0, d_1 and d_2 satisfy this property.

Next, we want to derive procedures that given a pair of languages compute these distances. Naturally, this is straighforward in case the input languages are finite. This task becomes more interesting when infinite languages are involved. For this, we would need to first have a finite representation of the input languages and then some machinery operating on this representation. In Section 3.2 we shortly examined how regular languages can be represented by finite automata. While we will stick to regular languages, finite automata no longer suffice, since we would prefer a single object representing the pair of languages. This is the objective of the next section.

3.4 Finitely representing tuples of languages

In this section, we demonstrate how tuples of (possibly infinite) languages can be (finitely) represented. This is motivated by our need to obtain a method for computing distances between languages, but it will also find use in other applications, like in Section 8.2.

Clearly, as was the case for languages, it is not possible to finitely represent all tuples of languages. Since a finite automaton can be used to represent a language, a natural idea is to use *tuples* of finite automata to represent tuples of languages. However, this is not very convenient. This is only a first step towards our goal of computing distances between languages, and hence we would like to obtain a *single* object that is able to interact with other tools, in particular the weighted automata we will investigate in the next section. To this end, we will make a transition from word languages to trees.

Initially we will introduce infinite trees and describe how tuples of languages can be mapped onto such trees, following ideas in [BN01; BO13; Pen15]. Then we will present tree automata, which like finite (word) automata are used to represent (families of) trees. Finally, based on the relation between languages and trees, we will investigate the connection between tuples of finite automata and tree automata.

Definition 3.7. *Let $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ be a non-empty finite set of symbols. Given a set of labels L, an L-labeled Σ-tree is a mapping $t : \Sigma^* \to L$ that assigns a label $t(w) \in L$ to every node $w \in \Sigma^*$. The set of all L-labeled Σ-trees is denoted as $T^{\omega}_{\Sigma,L}$. A set of trees $T \subseteq T^{\omega}_{\Sigma,L}$ is called a* tree language. *A* path *π of t is a subset of Σ^* such that the root belongs to π, and for every $w \in \pi$ exactly one of $w\sigma_1, \dots, w\sigma_k$ belongs to π.* ♢

Intuitively, the nodes of a Σ-tree t correspond to (finite) words in Σ^*, where the empty word ε represents the root of t and every node w has k children corresponding to the words $w\sigma_1, \dots, w\sigma_k$. Since for a non-empty alphabet Σ the set Σ^* of all words over Σ is infinite, Σ-trees are by definition infinite.

To represent tuples of languages we will employ infinite trees that use tuples over $\{0,1\}$ to label their nodes. We next define such a correspondence.

Definition 3.8. *Let Σ be a finite set of symbols and $\ell \in \mathbb{N}$. We define the mapping $\gamma_\ell : \left(2^{\Sigma^*}\right)^\ell \to T^{\omega}_{\Sigma,\{0,1\}^\ell}$ as follows. Given a tuple of languages $\mathbf{M} = (M_1, \dots, M_\ell)$ over Σ, $\gamma_\ell(\mathbf{M}) := t_{\mathbf{M}}$ where $t_{\mathbf{M}} : \Sigma^* \to \{0,1\}^\ell$ is the Σ-tree such that*

$$t_{\mathbf{M}}(w) := (x_1, \dots, x_\ell), \text{ where } x_i = 1 \text{ iff } w \in M_i \quad (\text{for all } w \in \Sigma^*).$$

♢

It is easy to see that γ_ℓ is a *bijection* between tuples of languages over the alphabet Σ and $\{0,1\}^\ell$-labeled Σ-trees. Given a tree $t \in T^{\omega}_{\Sigma,\{0,1\}^\ell}$, the inverse function yields the tuple $\gamma_\ell^{-1}(t) = (M_1, \dots, M_\ell)$ where M_i consists of the words w for which the ith component of $t(w)$ is equal to 1.

Example 3.9. *Consider the languages given by regular expressions $K = r^* \cup sr^*$ and $M = ss^*$. To express the tuple (K, M) as a tree, we assume that r is the first symbol of the alphabet and s is the second. Then this tuple is represented by the tree sketched on Figure 3.10. For better readability, we have labeled the edges with the symbols r and s. As an example for the labeling, consider the node corresponding to the word sr. It has label $(1,0)$ since this word belongs to $r^* \cup sr^*$, but not to ss^*. The extension of this tree to infinity is obtained as follows. On the one hand, the outgoing* dotted *edges tell us that all the nodes below are labeled with the tuple $(0,0)$. Notice, for example, that there are no words starting with rs or srs in any of the two languages.*

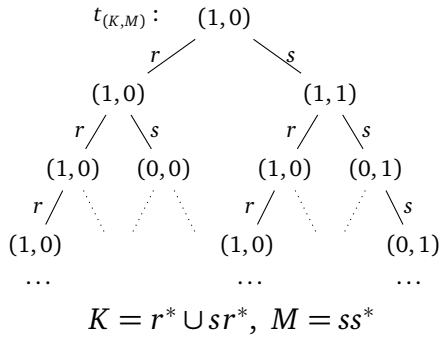

Figure 3.10: Tuple of languages as infinite tree.

On the other hand, the nodes rrr, srr and sss are the roots of infinite trees representing the tuples of languages $(r^*, \emptyset)$, $(r^*, \emptyset)$ and $(\emptyset, s^*)$, respectively. ♢

As mentioned before, our goal is to represent such tuples in a finite way. Using infinite trees obviously does not solve this problem. Thus, we need to develop an approach for representing such trees in a finite way. For general tuples of infinite languages and thus arbitrary Σ-trees this is clearly not possible. However, for the scope of our results, we can restrict our attention to the class of *regular* trees, which admit a finite representation.

We start by formally defining the notion of a regular tree, and then show that regular trees can always be represented using certain kinds of tree automata.

Definition 3.11 (regular tree). *Let t be a tree in $T^{\omega}_{\Sigma,L}$. Given a node $w \in \Sigma^*$, the* subtree *$t_w : \Sigma^* \to L$ of t is defined as $t_w(v) := t(wv)$ for all $v \in \Sigma^*$. We say that t contains the subtree t' if there exists $w \in \Sigma^*$ such that $t' = t_w$. Then, t is a* regular *tree if it contains finitely many distinct subtrees.* ♢

Reference [Cou83] is a survey which covers the basic theory and applications of regular trees. From the different ways that exist to represent regular trees in a finite way [Tho90], here, we use tree automata. A tree automaton starts its computation by assigning an initial state to the root and then works down to the rest of the tree. The transition relation specifies which states can be assigned to the children of a node, given the state and the label of the node. The tree automaton accepts the tree if there is a run built up in this fashion which is "successful", a notion that depends on the particular model of tree automata. A basic such model is *Büchi tree automata*.

Definition 3.12. *A* Büchi tree automaton (BTA) *is a tuple $\mathcal{A} = (\Sigma, Q, L, \Delta, I, F)$ where $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is a finite set of symbols, Q is a finite set of states, L is a finite set of labels, $\Delta \subseteq Q \times L \times Q^k$ is the transition relation, $I \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of final states. A run of $\mathcal{A}$ on a tree $t \in T^{\omega}_{\Sigma,L}$ is a Q-labeled Σ-tree $r : \Sigma^* \to Q$ such that $r(\varepsilon) \in I$ and*

$$(r(w), t(w), r(w\sigma_1), \ldots, r(w\sigma_k)) \in \Delta$$

for all $w \in \Sigma^*$. The run r of $\mathcal{A}$ on t is called successful *if for every path π of t we have*

$$\{q \in Q \mid r(w) = q \text{ for infinitely many } w \in \pi\} \cap F \neq \emptyset.$$

The tree language $\mathcal{L}(\mathcal{A})$ recognized *by* $\mathcal{A}$ *is the set of all trees* $t \in T^{\omega}_{\Sigma,L}$ *such that* $\mathcal{A}$ *accepts* t*, i.e.,* $\mathcal{A}$ *has a successful run on* t. ♢

The following result is due to Rabin [Rab72][5].

Theorem 3.13. *Any nonempty Büchi recognizable set of trees contains a regular tree.*

An important decision problem for a class of automata is testing for *emptiness*, that is, given an automaton $\mathcal{A}$ in that class, check whether there is any tree accepted by $\mathcal{A}$. Vardi and Wolper [VW86] proved that the emptiness test for Büchi tree automata is P-complete.

For the purposes of this chapter, a special case of Büchi automata will suffice. In particular, we consider automata with a trivial accepting condition, called *looping tree automata*, that were introduced in [VW94].

Definition 3.14. *A* looping tree automaton (LTA) *is a Büchi tree automaton* $\mathcal{A} = (\Sigma, Q, L, \Delta, I, F)$ *with* $F = Q$*. This implies that every run of* $\mathcal{A}$ *is successful, and hence a tree* t *is accepted by* $\mathcal{A}$ *iff* $\mathcal{A}$ *has a run on* t*. For this reason, when talking about an LTA we will omit* F. ♢

Testing emptiness of this restricted class of tree automata was proved to be possible in time linear in the size of the automaton [BT01].

In general, LTAs (and obviously also BTAs) recognize *sets* of trees. Therefore, to uniquely represent a tree we only consider those recognizing singleton sets.

Definition 3.15. *Let* $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$ *be a looping tree automaton. We say that* $\mathcal{A}$ represents the infinite tree $t \in T^{\omega}_{\Sigma,L}$ *if* $\mathcal{L}(\mathcal{A}) = \{t\}$. ♢

It is easy to see that trees that can be represented by looping tree automata are indeed regular. In fact, since LTAs are (a special case of) BTAs, by Theorem 3.13 we obtain that if such an automaton recognizes the singleton set $\{t\}$, then t must be regular. Conversely, we can show that any regular tree can be represented in this way.

Proposition 3.16. *Let* $t \in T^{\omega}_{\Sigma,L}$ *be an* L*-labeled* Σ*-tree. Then,* t *is regular iff it can be represented by an LTA.* ♢

Proof. We have already seen that the if-direction holds. To show the only-if direction, assume that t is a regular tree. By Definition 3.11 it thus contains only finitely many distinct subtrees, say $t^0, t^1, \ldots, t^m$ where we assume without loss of generality that $t^0 = t$. For all $1 \leq i \leq m$, we denote the direct subtrees of t^i as $t^i_{\sigma_1}, \ldots, t^i_{\sigma_k}$. Note that these are also subtrees of t, and thus belong to the set $\{t^0, t^1, \ldots, t^m\}$. We build the looping tree automaton $\mathcal{A}_t = (\Sigma, Q_t, L, \Delta_t, \{t^0\})$ as follows: we set $Q_t := \{t^0, t^1, \ldots, t^m\}$ and $\Delta_t := \{(t^i, t^i(\varepsilon), t^i_{\sigma_1}, \ldots, t^i_{\sigma_k}) \mid 1 \leq i \leq m\}$.

In the following, we show that that $t = t^0$ is the only tree accepted by $\mathcal{A}_t$. Initially, we prove that $\mathcal{A}_t$ accepts t, by inductively defining a run r of $\mathcal{A}_t$ on t. Set $r(\varepsilon) = t^0 = t_\varepsilon$. Assume that for $w \in \Sigma^*$, the state $r(w)$ has already been defined and it holds that $r(w) = t_w = t^j$ for

[5]The actual theorem in [Rab72] is about *Rabin* tree automata, which is a generalization of BTAs. This setting is too general for our purposes, but the result holds nevertheless.

some $0 \leq j \leq m$. Note that $t^j_{\sigma_i} = t_{w\sigma_i}$ for every $1 \leq i \leq k$. Since $t(w) = t_w(\varepsilon) = t^j(\varepsilon)$ and $(t^j, t^j(\varepsilon), t^j_{\sigma_1}, \ldots, t^j_{\sigma_k}) \in \Delta$, we define $r(w\sigma_i) = t^j_{\sigma_i} = t_{w\sigma_i}$. In this way, the run r of $\mathcal{A}_t$ on t is inductively defined, and thus $t \in \mathcal{L}(\mathcal{A}_t)$.

Next, assume that $\mathcal{A}_t$ has a run r' on a tree t'. We will inductively show that $r' = r$ and $t' = t$. Note that $r'(\varepsilon) = t_0 = r(\varepsilon)$, since otherwise r' would not be a run. The induction hypothesis is that $r'(w) = r(w) = t^j$ for some $0 \leq j \leq m$. Recall that, by construction of r, $r(w) = t_w$. Hence by construction of $\mathcal{A}_t$, $(r'(w), t'(w), r'(w\sigma_1), \ldots, r'(w\sigma_k)) \in \Delta$ implies that this tuple coincides with $(t^j, t^j(\varepsilon), t^j_{\sigma_1}, \ldots, t^j_{\sigma_k})$, i.e., $t'(w) = t^j(\varepsilon) = t_w(\varepsilon) = t(w)$ and furthermore $(r'(w\sigma_1), \ldots, r'(w\sigma_k)) = (t^j_{\sigma_1}, \ldots, t^j_{\sigma_k}) = (r(w\sigma_1), \ldots, r(w\sigma_k))$. Thus, for every $w \in \Sigma^*$ we have that $r'(w) = r(w)$ and $t'(w) = t(w)$ and hence $r' = r$ and $t' = t$, implying that $\mathcal{A}_t$ accepts exactly t and has a unique run on it. □

The automaton $\mathcal{A}_t$ constructed in the above proof actually has a very specific syntactic shape (see Definition 3.17 below), which ensures that it accepts only one tree.

Definition 3.17 (Representing looping tree automaton (rLTA)). *A* representing *looping tree automaton is a looping tree automaton $\mathcal{A} = (\Sigma, P, L, \Delta, \{p_s\})$ such that Δ satisfies the following condition:*

- *for every $p \in P$, there exists a unique symbol $l_p \in L$ and a unique tuple $(p_1, \ldots, p_{|\Sigma|}) \in P^{|\Sigma|}$ such that $(p, l_p, p_1, \ldots, p_{|\Sigma|}) \in \Delta$.* ♢

The following proposition states some obvious consequences of this definition and the proof of Proposition 3.16.

Proposition 3.18. *Let $\mathcal{A}$ be an rLTA and t a regular tree. Then*

1. *t can be represented by some rLTA $\mathcal{A}_t$.*
2. *$L(\mathcal{A})$ is a singleton set consisting of a regular tree $t_\mathcal{A}$ and $\mathcal{A}$ has a unique run $r_\mathcal{A}$ on $t_\mathcal{A}$.* ♢

Proof. The first claim is immediate after observing that the automaton $\mathcal{A}_t$ introduced in the proof of Proposition 3.16 is an rLTA. For the second claim, completely analogously to the proof of Proposition 3.16, we can prove that $\mathcal{A}$ has a run r on some tree t, and for any run r' of $\mathcal{A}$ on some tree t' it holds that $r' = r$ and $t' = t$. □

In case we are given a general LTA $\mathcal{A}$, we should like to know whether it actually represents a tree (i.e., recognizes a singleton set), and if the answer is affirmative construct an rLTA that represents the same tree.

Lemma 3.19. *Let $\mathcal{A}$ be an LTA. We can decide in polynomial time whether $\mathcal{A}$ represents a tree. If $\mathcal{A}$ represents a tree $t \in T^\omega_{\Sigma,L}$, then we can construct an rLTA representing t in polynomial time.*

Proof. Given $\mathcal{A}$, we remove superfluous states by applying the emptiness test for looping tree automata [BT01; BO13] and check whether $\mathcal{L}(\mathcal{A}) = \emptyset$. If this is the case, $\mathcal{A}$ does not represent a tree. Otherwise, we check whether the automaton accepts a unique tree. If the answer is affirmative, we obtain an automaton $\mathcal{A}^r$ by removing all but one transition for

every state. Obviously, $\mathcal{A}^r$ is an rLTA and $\mathcal{L}(\mathcal{A}^r) \subseteq \mathcal{L}(\mathcal{A})$. If $\mathcal{A}$ represents a tree, $\mathcal{A}^r$ is the rLTA we are looking for.

Before providing the exact algorithm below, a definition is due. Claiming that an LTA has no superfluous states is the informal way of saying that the LTA is trim. An LTA $\mathcal{A}$ is called *trim* if every state can be used in some run of $\mathcal{A}$. It is easy to see that every LTA can be transformed into a trim LTA that is equivalent in the sense of having the same runs.

Algorithm for deciding whether a given LTA represents a tree.
Given an LTA $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$:

- Construct an equivalent trim LTA $\mathcal{A}' = (\Sigma, Q', L, \Delta', I')$ [BO13, Lemma 2]. If the resulting automaton has no initial states, then $L(\mathcal{A}) = \emptyset$, and thus $\mathcal{A}$ does not represent a tree.
- Otherwise, compute the binary relation $\sim$ on Q' (that is inspired from automata minimization) as follows:
  - $B_0 = \{(q, q') \in Q'^2 \mid \exists (q, \ell_1, \ldots), (q', \ell_2, \ldots) \in \Delta' \text{ with } \ell_1 \neq \ell_2\}$
  - For $i = 1, 2, \ldots$, set

$$B_i = B_{i-1} \cup \{(q, q') \in Q'^2 \mid \exists (q, \ell, q_1, \ldots, q_k), (q', \ell, q'_1, \ldots, q'_k) \in \Delta' \text{ and } 1 \leq i \leq k \text{ s.t. } (q_i, q'_i) \in B_{i-1}\}.$$

  The iteration becomes stable and thus terminates after $m \leq |Q'^2|$ steps. Define $\sim := Q'^2 \setminus B_m$.
- Check whether $q \sim q'$ for every $q, q' \in I'$. The answer is positive iff $\mathcal{A}$ represents a tree.

The following lemma proves correctness of the algorithm.

Lemma 3.20. *A trim LTA $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$ represents a tree iff $I \neq \emptyset$ and $q \sim q'$ for every $q, q' \in I$.*

Proof. Assume that $\mathcal{A}$ does not represent a tree. This means that either it does not accept any tree, or it accepts more than one. In the first case, since $\mathcal{A}$ is trim, we get that $I = \emptyset$. In the second case, there are at least two trees t_1, t_2 accepted by $\mathcal{A}$. Let $w = \sigma_{i_1} \ldots \sigma_{i_n}$ be a minimal word s.t. $t_1(w) \neq t_2(w)$ and r_1, r_2 be runs of $\mathcal{A}$ on t_1, t_2 respectively. Thus, we get that there are transitions $(r_1(w), t_1(w), \ldots), (r_2(w), t_2(w), \ldots) \in \Delta$ with $t_1(w) \neq t_2(w)$. By the construction in the algorithm, $(r_1(w), r_2(w)) \in B_0 \subseteq B_m$. For every proper prefix v of w, since $t_1(v) = t_2(v)$ (by minimality of w) we get that $(r_1(v), r_2(v)) \in B_m$. In particular, $(r_1(\varepsilon), r_2(\varepsilon)) \in B_m$, and since $r_1(\varepsilon), r_2(\varepsilon) \in I$ the proof of this direction is complete.

For the other direction, if $I = \emptyset$ then obviously $\mathcal{A}$ does not accept any trees. Assume that $q \not\sim q'$, i.e., $(q, q') \in B_m$ for some $q, q' \in I$. Then, let l be the least number such that $(q, q') \in B_l$. If $l = 0$, there exist $(q, \sigma, \ldots), (q', \sigma', \ldots) \in \Delta$ with $\sigma \neq \sigma'$, and since the automaton is trim, we get that $\mathcal{A}$ accepts at least two trees, one with root σ and one with σ'. If $l \geq 1$, there exist $(q, \sigma, q_1, \ldots, q_k), (q', \sigma, q'_1, \ldots, q'_k) \in \Delta$ with $(q_i, q'_i) \in B_{l-1}$ for some $1 \leq i \leq k$. Iterating the above argument, we get a word $w \in \Sigma^*$ (with length at most l) and a pair $(p, p') \in B_0$ s.t. p is a w-successor of q and p' of q' and, as before, we derive that $\mathcal{A}$ accepts at least two trees (with the difference existing in the node w instead of the root). $\square$

The results of this section show that we can restrict the attention to rLTAs when representing regular trees.

Finally, we are ready to implement our original idea: we show how to obtain a single object, i.e., an rLTA, from a tuple of DFAs representing a tuple of regular languages.

Lemma 3.21. *Given a tuple of DFAs $\mathbf{A} = (A_1, \dots, A_k)$, each of size at most n, recognizing the tuple of languages $\mathbf{L} = (L_1, \dots, L_k)$, we can construct an rLTA $\mathcal{A}_\mathbf{A}$ of size n^k representing $t_\mathbf{L} = \gamma_k(\mathbf{L})$.*

Proof. Assume that $\Sigma = \{\sigma_1, \dots, \sigma_m\}$ and the DFAs are of the form $A_i = (Q_i, \Sigma, q_i^0, \delta_i, F_i)$. We define $\mathcal{A}_\mathbf{A} := (P, \Sigma, L, \Delta, \{p^0\})$, where $P := Q_1 \times \cdots \times Q_k$, $L := \{0,1\}^k$, $p^0 := (q_1^0, \dots, q_k^0)$, and

$$\Delta := \{(p, \ell, p_1, \dots, p_n) \mid p = (q_1, \dots, q_k) \in P,\ \ell = (x_1, \dots, x_k),\ x_i = 1 \text{ iff } q_i \in F_i,\\ p_i = (\delta_1(q_1, \sigma_i), \dots, \delta_k(q_k, \sigma_i)) \text{ for } i = 1, \dots, m \quad \}.$$

Initially, note that by construction $\mathcal{A}_\mathbf{A}$ is an rLTA. Hence, it has exactly one run, say r, on exactly one tree, say t. We will now prove that $t = t_\mathbf{L}$, that is, for every $w \in \Sigma^*$ it holds that $t(w) = (x_1, \dots, x_k)$ with $x_i = 1$ iff $w \in L_i$.

By induction, it is easy to see that for every $w \in \Sigma^*$ holds $r(w) = (\delta_1^*(q_1^0, w), \dots, \delta_k^*(q_k^0, w))$. Since for every state $p \in P$ there exists a unique label ℓ such that $(p, \ell, \dots) \in \Delta$, and $\mathcal{A}_\mathbf{A}$ has a run on t, we necessarily have that $t(w) = (x_1, \dots, x_k)$ with $x_i = 1$ iff $\delta_i^*(q_i^0, w) \in F_i$, which is exactly iff $w \in L_i$. □

If our starting point was a tuple of NFAs, this product construction would not have worked, for it heavily relies on the fact that the component automata work synchronously. Since rLTAs are by nature deterministic, in a sense they are "not compatible" with NFAs. In order to obtain a similar result, we would have to first obtain a tuple of equivalent DFAs and then perform the above construction. The entire procedure would, however, require time exponential in the size of the input NFAs.

In later chapters, it will also be useful to define (tuples of) languages from trees with an arbitrary set of labels. The following definition provides such a procedure.

Definition 3.22. *Given an L-labeled Σ-tree t the language* induced by the set $F \subseteq L$ *is defined to be $K_t(F) := \{w \in \Sigma^* \mid t(w) \in F\}$. Furthermore, the tuple of languages induced by the tuple of sets $F_1, \dots, F_k \subseteq L$ is defined to be $K_t(F_1, \dots, K_k) := (K_t(F_1), \dots, K_t(F_k))$.* ♢

3.5 Towards computing language distances

Our goal is now to assign values to tuples of (possibly infinite) languages that can be represented by regular trees. Consequently, we need a device that takes as input such a tree and returns a value. Weighted looping tree automata are such devices: they assign values (from a so-called semiring) to infinite trees. In the next subsection, we introduce the special type of weighted tree automata that we will use together with the necessary notions (semirings, discounting, etc.). We will then show how the language distances d_0, d_1, d_2 introduced in Section 3.3 can be realized using such automata.

3.5.1 Weighted looping tree automata

In order to assign a value to a tree, weighted tree automata make use of transitions that are equipped with weights. These weights are usually elements of a semiring such that one can add and multiply weights. An extensive survey of weighted tree automata can be found in [FV09]. In a setting where the automata are required to work on infinite trees, the underlying semiring should admit suitable infinite sums and products [Rah07]. In the context of infinite trees, it is also useful to employ discounting. This has been used for modeling systems with non-terminating behavior [AHM03] in order to assign different degrees of importance to incidents that happen later in time. In our setting, discounting can be used to assign less importance to differences that occur for longer words, i.e., further down in the tree.

Semirings.

The weight structures underlying our weighted tree automata are totally complete commutative semirings [Rah07].

Definition 3.23. *A* semiring $\mathcal{S} = (S, \oplus, \otimes, 0, 1)$ *consists of a set* S*, two binary operations* $\oplus$ *and* $\otimes$*, and two constant elements* 0 *and* 1 *such that:*

1. $(S, \oplus, 0)$ *is a commutative monoid,*
2. $(S, \otimes, 1)$ *is a monoid,*
3. *multiplication distributes over addition from left and right,*
4. $0 \otimes a = a \otimes 0 = 0$ *for all* $a \in S$.

A semiring is called commutative *if* $a \otimes b = b \otimes a$ *for all* $a, b \in S$. ♢

Next, assume that addition can be suitably extended to infinite sums, i.e., the semiring $\mathcal{S}$ is equipped with infinitary sum operations $\bigoplus_I : S^I \to S$, for any index set I, such that for all I and all families $(a_i \mid i \in I)$ of elements of S the following hold:

$$\bigoplus_{i \in \emptyset} a_i = 0, \qquad \bigoplus_{i \in \{j\}} a_i = a_j, \qquad \bigoplus_{i \in \{j,k\}} a_i = a_j \oplus a_k \text{ for } j \neq k,$$

$$\bigoplus_{j \in J} \left(\bigoplus_{i \in I_j} a_i \right) = \bigoplus_{i \in I} a_i, \text{ if } \bigcup_{j \in J} I_j = I \text{ and } I_j \cap I_k = \emptyset \text{ for } j \neq k,$$

$$\bigoplus_{i \in I} (c \otimes a_i) = c \otimes \left(\bigoplus_{i \in I} a_i \right), \qquad \bigoplus_{i \in I} (a_i \otimes c) = \left(\bigoplus_{i \in I} a_i \right) \otimes c.$$

The semiring $\mathcal{S}$ together with the operations $\bigoplus_I$ is called *complete*.

A complete semiring is said to be *totally complete*, if it is endowed with countably infinite product operations satisfying for all sequences $(a_i \mid i \geq 0)$ of elements of S the following conditions:

$$\bigotimes_{i \geq 0} 1 = 1, \quad a_0 \otimes \bigotimes_{i \geq 0} a_{i+1} = \bigotimes_{i \geq 0} a_i, \quad \bigotimes_{i \geq 0} a_i = \bigotimes_{i \geq 0} a_i',$$

where $a_0{}' = a_0 \otimes \ldots \otimes a_{n_1}$, $a_1{}' = a_{n_1+1} \otimes \ldots \otimes a_{n_2}$, $\ldots$ for an increasing sequence of natural numbers $0 < n_1 < n_2 < \ldots$, and

$$\bigotimes_{j\geq 1}\left(\bigoplus_{i\in I_j} a_i\right) = \bigoplus_{(i_1,i_2,\ldots)\in I_1\times I_2\times\ldots}\left(\bigotimes_{j\geq 1} a_{i_j}\right),$$

where $I_1, I_2, \ldots$ are arbitrary index sets.

A *totally commutative complete semiring* is a commutative and totally complete semiring that additionally satisfies:

$$\bigotimes_{i\geq 0}(a_i \otimes b_i) = \left(\bigotimes_{i\geq 0} a_i\right) \otimes \left(\bigotimes_{i\geq 0} b_i\right).$$

Examples The following semirings are totally commutative complete:

- the semiring $(\mathbb{N}\cup\{+\infty\}, +, \cdot, 0, 1)$ of natural numbers extended with positive infinity $+\infty$,
- the *tropical semiring* $Trop = (\mathbb{N}\cup\{+\infty\}, \min, +, +\infty, 0)$ and the *arctic semiring* $Arc = (\mathbf{N}\cup\{+\infty, -\infty\}, \sup, +, -\infty, 0)$ with the binary operations extended in the natural way to infinitary operations,
- the counterparts of the aforementioned semirings over the nonnegative real numbers, $\mathbb{R}_{\inf} = (\mathbb{R}_{\geq 0}\cup\{+\infty\}, \inf, +, +\infty, 0)$ and $\mathbb{R}_{\sup} = (\mathbb{R}_{\geq 0}\cup\{+\infty, -\infty\}, \sup, +, -\infty, 0)$,
- the *Viterbi* semiring $([0,1], \sup, \cdot, 0, 1)$,
- every complete distributive lattice.

All of the above examples but Viterbi can be found in [Rah07]. To the best of our knowledge, whether the Viterbi semiring is totally commutative complete has not been investigated in the literature before. To prove that this is indeed the case, it suffices to make the following two observations:

- It is well-known (see for example [Kno51]) that the infinite product $\prod_{i\geq 0} a_i$ converges in case $\sum_{i\geq 0}(1-a_i)$ converges, and is equal to 0 if $\sum_{i\geq 0}(1-a_i) = +\infty$. Since $0 \leq a_i \leq 1$, we have that $\sum_{i\geq 0}(1-a_i)$ either converges or is equal to $+\infty$, and thus the infinite product is well-defined.
- For any index set I and any family $(a_i \mid i \in I)$ of elements in $[0,1]$ it holds that

  $$\sup_{i\in I} a_i = e^{-\inf_{i\in I}\{-\log a_i\}},$$

  and for any sequence $(a_i \mid i \geq 0)$ of elements in $[0,1]$

  $$\prod_{i\geq 0} a_i = e^{-\sum_{i\geq 0}(-\log a_i)}.$$

Thus, Viterbi being a totally commutative complete semiring is a corollary of $\mathbb{R}_{\inf}$ being one.

Discounting.

In the setting of semirings, discounting is defined by using semiring endomorphisms. This approach was originally used for weighted automata on infinite words by Droste and Kuske in [DK06], and extended to weighted automata on infinite trees by Mandrali and Rahonis [MR09].

Definition 3.24. *Let $\mathcal{S} = (S, \oplus, \otimes, 0, 1)$ be a semiring. A mapping $f : S \to S$ is called an* endomorphism *if $f(a \oplus b) = f(a) \oplus f(b)$ and $f(a \otimes b) = f(a) \otimes f(b)$ for all $a, b \in S$, and $f(0) = 0$, $f(1) = 1$. The set $End(\mathcal{S})$ of all endomorphisms of $\mathcal{S}$ is a monoid with composition $\circ$ as binary operation and the identity mapping id as unit.* ♢

For $\mathbb{R}_{\text{sup}}$, it was proved in [DK06] that every endomorphism is of the form $\overline{p}(a) = p \cdot a$ for some $p \in [0, +\infty)$, and conversely, every $p \in [0, +\infty)$ defines an endomorphism of $\mathbb{R}_{\text{sup}}$ in this way. The same result can be shown for $\mathbb{R}_{\text{inf}}$ as well [DR09]. Finally, it is not difficult to see that a similar result holds for the Viterbi semiring.

Lemma 3.25. *In the Viterbi semiring $([0, 1], \sup, \cdot, 0, 1)$, every endomorphism is of the form $\tilde{p}(a) = a^p$ for some $p \in [0, +\infty)$, and conversely every $p \in [0, +\infty)$ defines an endomorphism of Viterbi.*

Proof. Initially, observe that for every $a, b \in [0, 1]$ and for every $p \in [0, +\infty)$ it holds that

- $\tilde{p}(\sup\{a, b\}) = (\sup\{a, b\})^p = \sup\{a^p, b^p\} = \sup\{\tilde{p}(a), \tilde{p}(b)\}$,
- $\tilde{p}(a \cdot b) = (a \cdot b)^p = a^p \cdot b^p = \tilde{p}(a) \cdot \tilde{p}(b)$,
- $\tilde{p}(0) = 0^p = 0$ and
- $\tilde{p}(1) = 1^p = 1$.

Thus, every $p \in [0, +\infty)$ defines an endomorphism of Viterbi.

Next, assume that Φ is an endomorphism of Viterbi, and define

$$\phi(x) = -\log(\Phi(e^{-x})).$$

We will show that ϕ is an endomorphism of $\mathbb{R}_{\text{inf}}$. Indeed, for every $x, y \in \mathbb{R}_{\text{inf}}$ it holds that

$$\begin{aligned}\phi(\inf\{x, y\}) &= -\log(\Phi(e^{-\inf\{x,y\}})) = -\log(\Phi(\sup\{e^{-x}, e^{-y}\})) \\ &= -\log(\sup\{\Phi(e^{-x}), \Phi(e^{-y})\}) = \inf\{-\log(\Phi(e^{-x})), -\log(\Phi(e^{-y}))\} \\ &= \inf\{\phi(x), \phi(y)\}\end{aligned}$$

and also

$$\begin{aligned}\phi(x + y) &= -\log(\Phi(e^{-(x+y)})) = -\log(\Phi(e^{-x} \cdot e^{-y})) \\ &= -\log(\Phi(e^{-x}) \cdot \Phi(e^{-y})) = -\log(\Phi(e^{-x})) - \log(\Phi(e^{-y})) \\ &= \phi(x) + \phi(y).\end{aligned}$$

Thus, ϕ is indeed an endomorphism of $\mathbb{R}_{\inf}$. Consequently, by [DR09], we know that there exists some $p \in [0, +\infty)$, such that $\phi(x) = p \cdot x$ for every $x \in [0, +\infty)$. Hence, for every $x \in [0, +\infty)$ we get that

$$\phi(x) = p \cdot x \implies -\log(\Phi(e^{-x})) = p \cdot x \implies \Phi(e^{-x}) = e^{-p \cdot x} = (e^{-x})^p.$$

Since the image of the interval $[0, +\infty)$ under the function e^{-x} is the interval $(0, 1]$, for every $a \in (0, 1]$ we have that $\Phi(a) = a^p$. Finally, since by definition of endomorphism $\Phi(0) = 0 = 0^p$, we get that $\Phi(a) = a^p$ holds in the complete interval, i.e., for every $a \in [0, 1]$. □

Definition 3.26. *Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a finite set of symbols and $\mathcal{S}$ a semiring. A* discounting *for Σ and $\mathcal{S}$ is a tuple $\Phi \in (End(\mathcal{S}))^k$.*[6] ♢

For a discounting $\Phi = (\phi_1, \ldots, \phi_k)$ and for every word $w = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_n} \in \Sigma^*$, we define the endomorphism ϕ_w of $\mathcal{S}$ induced by Φ and w as $\phi_w = \phi_{i_1} \circ \phi_{i_2} \circ \cdots \circ \phi_{i_n}$, where for $w = \varepsilon$ the empty composition is *id*.

Weighted looping tree automata.

In the following, $\mathcal{S}$ is assumed to be a totally complete commutative semiring. An *infinitary tree series* h over L and $\mathcal{S}$ is a mapping $h : T^{\omega}_{\Sigma,L} \to S$. The class of all infinitary tree series over L and $\mathcal{S}$ is denoted by $\mathcal{S}\langle\langle T^{\omega}_{\Sigma,L}\rangle\rangle$.

Definition 3.27 (Weigthed looping tree automaton with discounting Φ). *A* weighted looping tree automaton with discounting Φ *(Φ-wLTA) over $\mathcal{S}$ is a tuple $\mathcal{M} = (\Sigma, Q, L, in, wt)$ where Q is a finite state set, L is a finite set of labels, $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is a finite set of symbols, $in : Q \to S$ is the initial distribution, and $wt : Q \times L \times Q^k \to S$ is a mapping assigning weights to the transitions of the automaton.* ♢

Given a Φ-wLTA $\mathcal{M} = (\Sigma, Q, L, in, wt)$ over $\mathcal{S}$, a *run* of $\mathcal{M}$ on a tree $t \in T^{\omega}_{\Sigma,L}$ is a mapping $r : \Sigma^* \to Q$. We denote the set of all runs of $\mathcal{M}$ on t by $R_{\mathcal{M}}(t)$. Given a run r, we denote the transition $(r(w), t(w), r(w\sigma_1), \ldots, r(w\sigma_k))$ by $\overrightarrow{r}(w)$. The *weight* of the run r at $w \in \Sigma^*$ is defined as $wt(r, w) := wt(\overrightarrow{r}(w))$. The *$\Phi$-weight* (or simply weight) of r is defined as

$$weight(r) := in(r(\varepsilon)) \otimes \bigotimes_{w \in \Sigma^*} \phi_w(wt(r, w)).$$

Finally, the Φ-behavior (or simply behavior) of $\mathcal{M}$ is the infinitary tree series $||\mathcal{M}|| \in \mathcal{S}\langle\langle T^{\omega}_{\Sigma,L}\rangle\rangle$ whose coefficients are determined for every $t \in T^{\omega}_{\Sigma,L}$ by

$$(||\mathcal{M}||, t) := \bigoplus_{r \in R_{\mathcal{M}}(t)} weight(r).$$

If we take $\phi_i = id$ for every $i = 1, \ldots, k$, then we are left with a "normal" wLTA over $\mathcal{S}$ in the sense of [Rah07], and thus dispense with the prefix Φ- in the notation.

[6] In the literature, more general forms of discounting have been introduced, where the tuple of endomorphisms to be used depends also on the label of a node, but here we restrict our attention to the simpler form of discounting introduced above.

If $|L| = 1$, then $T^{\omega}_{\Sigma,L}$ consists of a single tree t_{ul}, which we will call the *unlabeled tree* since the labels are then irrelevant. In this case, we omit the label from the transitions of a Φ-wLTA $\mathcal{M}$ and write $R_{\mathcal{M}}$ for its runs, omitting t_{ul}. Also note that then $\|\mathcal{M}\|$ is a single element of S rather than a tree series.

3.5.2 Expressing language distances

In this section we show that our approach of mapping languages to trees is actually useful, that is, wLTAs can be used to model language distances.

The functions d_0, d_1, d_2 introduced in Section 3.3 take a pair of languages over an alphabet Σ as input. Thus, to represent this kind of input in a tree, we use the label set $L_2 := \{0,1\}^2$. We show that d_0, d_2 as well as a vital component of d_1 can be expressed by weighted looping automata with discounting over $\mathbb{R}_{\text{inf}}$. The function d_1 itself can be expressed using the Viterbi semiring.

Example 3.28. *We start with the simplest language distance, $d_0(K,N) = |K \,\Delta\, N|$. We introduce a wLTA (without discounting) that, given a tree t representing the tuple of languages (K,N), computes $|K \,\Delta\, N|$. Consider the wLTA $\mathcal{M}_0 = (\Sigma, Q, L_2, in_0, wt_0)$ over $\mathbb{R}_{\text{inf}}$ where $Q = \{q_0, q_1\}$, $in_0(q_0) = in_0(q_1) = 0$ and*

$$wt_0(q, l, p_1, \ldots, p_k) = \begin{cases} 0 & \text{if } q = q_0,\ l \in \{(0,0),(1,1)\} \\ 1 & \text{if } q = q_1,\ l \in \{(1,0),(0,1)\} \\ +\infty & \text{otherwise} \end{cases}$$

It is easy to see that there is a unique run r_0 with non-infinite weight, the one that assigns q_0 to the nodes labeled with $(0,0)$ or $(1,1)$, i.e., words that do not belong to $K \,\Delta\, N$, and q_1 to the ones labeled with $(1,0)$ or $(0,1)$, i.e., words that belong to $K \,\Delta\, N$. If the word does not belong to $K \,\Delta\, N$ it gets a zero weight. If it does belong to $K \,\Delta\, N$, it gets weight 1. "Multiplying" in $\mathbb{R}_{\text{inf}}$, we add 1 for every word in $K \,\Delta\, N$, and hence we obtain exactly $|K \,\Delta\, N|$ as weight for this run.$\diamondsuit$

Example 3.29. *Next, we investigate the language distance d_1. Recall tbat $d_1(K,N) = 2^{-n}$ where $n = \min\{|w| \mid w \in K \,\Delta\, N\}$. We introduce a wLTA (without discounting) that, given a tree t representing the tuple of languages (K,N), computes the minimum n (rather than 2^{-n} itself). Given n, the exponentiation can be done by external computation. Consider the wLTA $\mathcal{M}_1 = (\Sigma, Q, L_2, in_1, wt_1)$ over $\mathbb{R}_{\text{inf}} = (\mathbb{R}_{\geq 0} \cup \{+\infty\}, \inf, +, +\infty, 0)$, where $Q = \{q_0, q_1\}$, $in_1(q_0) = +\infty$, $in_1(q_1) = 0$ and*

$$wt_1(q, l, p_1, \ldots, p_k) = \begin{cases} 1 & \text{if } q = q_1,\ l \in \{(0,0),(1,1)\},\ p_i = q_1 \text{ for some } 1 \leq i \leq k \\ & \qquad \text{and } p_j = q_0 \text{ for } j \neq i \\ 0 & \text{if } q = q_1,\ l \in \{(1,0),(0,1)\},\ p_i = q_0 \text{ for all } 1 \leq i \leq k \\ 0 & \text{if } q = q_0,\ l \in \{0,1\}^2,\ p_i = q_0 \text{ for all } 1 \leq i \leq k \\ +\infty & \text{otherwise} \end{cases}$$

Intuitively, each run using only transitions with non-infinite weights selects one path in the tree, which it labels with q_1 until an element in the symmetric difference is found. The transitions up to this point in the selected path receive weight 1, and all other transitions have weight 0. Thus, adding up the weights (with the multiplication $\otimes = +$ of $\mathbb{R}_{\text{inf}}$) gives us the distance from the

root to the node where the difference was detected, i.e., the length of the word in the symmetric difference (or $+\infty$ in case no difference is found on the chosen path). By building the infimum over all runs, the length of the shortest word in the symmetric difference is found.

Example 3.29'. *Actually, we can compute the exact value of d_1 by making use of the Viterbi semiring. Consider the wLTA $\mathcal{M}_1' = (\Sigma, Q, L_2, in_1', wt_1')$ over the Viterbi semiring $([0,1], \sup, \cdot, 0, 1)$ where $Q = \{q_0, q_1\}$, $in_1'(q_0) = 0$, $in_1'(q_1) = 1$ and*

$$wt_1'(q, l, p_1, \ldots, p_k) = \begin{cases} \frac{1}{2} & \text{if } q = q_1,\ l \in \{(0,0),(1,1)\},\ p_i = q_1 \text{ for some } 1 \le i \le k \\ & \qquad \text{and } p_j = q_0 \text{ for } j \neq i \\ 1 & \text{if } q = q_1,\ l \in \{(1,0),(0,1)\},\ p_i = q_0 \text{ for all } 1 \le i \le k \\ 1 & \text{if } q = q_0,\ l \in \{0,1\}^2,\ p_i = q_0 \text{ for all } 1 \le i \le k \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that this automaton works completely analogously to the previous one. Each run that does not use transitions with zero weight selects a path in the tree, which it labels with q_1 until an element in the symmetric difference is found. The transitions up to this point in the selected path receive weight $\frac{1}{2}$, and all other transitions have weight 1.

Thus, if the distance from the root to the node where the difference was detected is k, the weight of the run, i.e., the infinite product of the appearing weights (recall that $\otimes$ in Viterbi is actual multiplication), which contains mainly 1s and k $\frac{1}{2}$s is $(\frac{1}{2})^k$.

The supremum over all runs is obtained when the least $\frac{1}{2}$s appear, namely for the run that chooses the path that leads to the shortest word in the symmetric difference. Note that if no such word exists, for every such run the weight is a product containing infinitely many times $\frac{1}{2}$, and hence is 0. ♢

Example 3.30. *Finally, we take a look at d_2. Recall that $d_2(K,N) = \mu(K \,\Delta\, N)$, where $\mu(M) = \frac{1}{2}\sum_{w \in M}(2|\Sigma|)^{-|w|}$. We introduce a Φ-wLTA that, given a tree t representing the tuple of languages (K,N), computes $\mu(K \,\Delta\, N)$. Consider the Φ-wLTA $\mathcal{M}_2 = (\Sigma, Q, L_2, in_2, wt_2)$ over $\mathbb{R}_{\inf}$ where $Q = \{q_0, q_1\}$, $in_2(q_0) = in_2(q_1) = 0$ and*

$$wt_2(q, l, p_1, \ldots, p_k) = \begin{cases} 0 & \text{if } q = q_0,\ l \in \{(0,0),(1,1)\} \\ \frac{1}{2} & \text{if } q = q_1,\ l \in \{(1,0),(0,1)\} \\ +\infty & \text{otherwise} \end{cases}$$

Finally, the discounting $\Phi = (\phi_1, \ldots, \phi_k)$ is defined as $\phi_i = \overline{\frac{1}{2|\Sigma|}}$ for every $i = 1, \ldots, k$, where $\overline{\frac{1}{2|\Sigma|}}(a) = \frac{1}{2|\Sigma|} \cdot a$ for $a \in \mathbb{R}_{\ge 0}$ and $\overline{\frac{1}{2|\Sigma|}}(+\infty) = +\infty$.

It is easy to see that there is a unique run r_0 with non-infinite weight, the one that assigns q_0 to the nodes labeled with $(0,0)$ or $(1,1)$, i.e., words that do not belong to $K \,\Delta\, N$, and q_1 to the ones labeled with $(1,0)$ or $(0,1)$, i.e., words that belong to $K \,\Delta\, N$. The discounting multiplies the weight of every word $w \in \Sigma^*$ with $(\frac{1}{2|\Sigma|})^{|w|}$. If the word does not belong to $K \,\Delta\, N$ it gets a zero weight. If it does belong to $K \,\Delta\, N$, it gets weight $\frac{1}{2}$. "Multiplying" in $\mathbb{R}_{\inf}$ (i.e., summing over all words in Σ^*), we obtain exactly $\mu(K \,\Delta\, N)$ as weight for this run. ♢

Note that in all of the above examples, the automaton does not differentiate between the labels $(0,0)$ and $(1,1)$, and neither between $(1,0)$ and $(0,1)$. This is to be expected due to

the definition of the particular distances. More generally, for distances that are defined based on the symmetric difference (like, d_0, d_1 and d_2) we could also use an alternative approach: we could equivalently have trees representing a single language, the symmetric difference of the input languages, and have the weighted automaton operate on this tree. Obtaining weighted automata for the above distances that operate on such trees is straightforward given the above examples.

Overall, we say that a language distance d can be *expressed by* a (Φ)-wLTA $\mathcal{M}$ if $\mathcal{M}$ operates on $\{0,1\}^2$- (or $\{0,1\}$-labeled) trees, and for every pair of languages K, L it holds that the behavior of $\mathcal{M}$ on the tree representing the tuple (K, L) (or respectively the symmetric difference $K \Delta L$), coincides with the d-distance of K and L, in symbols $d(K, L) = (||\mathcal{M}||, t_{(K,L)})$ (or $d(K, L) = (||\mathcal{M}||, t_{K \Delta L})$ respectively).

3.5.3 Further considerations

Although we have been able to obtain wLTAs for the language distances we are investigating, it would still be interesting to futher explore this connection between language distances and wLTAs.

In particular, it is not clear if we can express distances like δ_2, which was shortly discussed in Section 3.3, using Φ-wLTAs. Most probably this is not the case for functions that involve limit computations[7]. Potentially, automata with weights over a so-called valuation monoid [DM10] could be of use. However, valuation functions usually operate like black boxes and it is not clear how (if) it is possible to compute the behavior of such automata, unlike Φ-wLTAs as we will see next.

More generally, we would ideally like to have a characterization of when a language distance can be expressed by a wLTA, and even a procedure to construct such an automaton. Conversely, we would like to have algorithms for checking whether a given Φ-wLTA expresses a language distance, that is, the function it defines on pairs of languages satisfies the properties $(M1)$-$(M3)$ of a metric, as well as syntactic restrictions on the automata that guarantee satisfaction of these properties.

This latter topic seems to have the best potential for fruitful investigation. For example, given a Φ-wLTA $\mathcal{M} = (\Sigma, Q, \{0,1\}^2, in, wt)$, for the property $(M1)$ it suffices to restrict wt such that for every $q \in Q$, $(q_1, \ldots, q_{|\Sigma|}) \in Q^{|\Sigma|}$

$$wt(q,(0,0),q_1,\ldots,q_{|\Sigma|}) = wt(q,(1,1),q_1,\ldots,q_{|\Sigma|}) = 0$$
$$wt(q,(0,1),q_1,\ldots,q_{|\Sigma|}) = wt(q,(1,0),q_1,\ldots,q_{|\Sigma|}) \geq 0.$$

Furthermore, for $(M2)$ it suffices to set for every $q \in Q$, $(q_1, \ldots, q_{|\Sigma|}) \in Q^{|\Sigma|}$

$$wt(q,(0,1),q_1,\ldots,q_{|\Sigma|}) = wt(q,(1,0),q_1,\ldots,q_{|\Sigma|}).$$

However, none of these two conditions are necessary, and furthermore, there is no obvious way to guarantee the triangle inequality $(M3)$ in a non-trivial way. Further investigation of the subject is an interesting topic for future research.

[7] cf. [KN06]

3.6 Computing the behavior of wLTAs on regular trees

Now, we turn to the problem of how to actually compute the value assigned by a wLTA to a regular tree. Formally, given a Φ-wLTA $\mathcal{M}$ over a semiring $\mathcal{S}$ and an rLTA $\mathcal{A}$ representing a regular tree t, we want to compute the behavior of $\mathcal{M}$ on t, i.e., $(||\mathcal{M}||, t)$. In a first step, we reduce this problem to the problem of computing the behavior of a Φ-wLTA on the unlabeled tree. To be more precise, we combine the two automata $\mathcal{M}$ and $\mathcal{A}$ into a single Φ-wLTA $\mathcal{M}_\mathcal{A}$ that works on the unlabeled tree t_{ul} such that $(||\mathcal{M}||, t) = (||\mathcal{M}_\mathcal{A}||, t_{ul})$. Notice that, since the unlabeled Σ-tree is unique, the behavior of the new wLTA is an *element* of the semiring and not a function. Thus, in order to compute $(||\mathcal{M}||, t)$ it suffices to be able to compute the behavior $\mathcal{M}_\mathcal{A}$; this will be the objective of the next section.

Theorem 3.31. *Given Φ-wLTA $\mathcal{M} = (\Sigma, Q, L, in, wt)$ over $\mathcal{S}$ and rLTA $\mathcal{A} = (\Sigma, P, L, \Delta, \{p_s\})$ representing a regular tree t, one can construct in polynomial time a Φ-wLTA $\mathcal{M}_\mathcal{A}$ over $\mathcal{S}$ working on the unlabeled tree t_{ul} such that $(||\mathcal{M}||, t) = (||\mathcal{M}_\mathcal{A}||, t_{ul})$.*

Proof. Let $\mathcal{S} = (S, \oplus, \otimes, 0, 1)$. By the definition of rLTAs, for every state $p \in P$ there exists a unique letter $l_p \in L$ such that $(p, l_p, \ldots) \in \Delta$. Additionally, by Proposition 3.18 it holds that $\mathcal{A}$ has a unique run, say θ, on t. For simplicity, for every $w \in \Sigma^*$ we denote $\theta(w) \in P$ by p_w.

We define the Φ-wLTA $\mathcal{M}_\mathcal{A} = (Q \times P \times L, \Sigma, in', wt')$ over $\mathcal{S}$ as follows:

$$in'(q,p,l) := \begin{cases} in(q) & \text{if } p = p_s \text{ and } l = l_{p_s} \\ 0 & \text{otherwise} \end{cases}$$

$$wt'\big((q_0,p_0,l_0),(q_1,p_1,l_1),\ldots,(q_k,p_k,l_k)\big) := \begin{cases} wt(q_0,l_0,q_1,\ldots,q_k) & \text{if } (p_0,l_0,p_1,\ldots,p_k) \in \Delta \\ 0 & \text{otherwise} \end{cases}$$

To prove that $(\|\mathcal{M}\|, t) = (\|\mathcal{M}_\mathcal{A}\|, t_{ul})$, it is sufficient to show that there exists an injection $\tau : R_\mathcal{M}(t) \to R_{\mathcal{M}_\mathcal{A}}$ such that $weight(r) = weight(\tau(r))$ for every $r \in R_\mathcal{M}(t)$ and $weight(r') = 0$ for every $r' \in R_{\mathcal{M}_\mathcal{A}} \setminus im(\tau)$, where $im(\tau)$ stands for the *image* set of the mapping τ.

More precisely, the injection is defined as follows. Given a run $r \in R_\mathcal{M}(t)$, we define $\tau(r) = r'$ by setting $r'(w) = (r(w), p_w, l_{p_w})$. We have $in'(r'(\varepsilon)) = in'(r(\varepsilon), p_\varepsilon, l_{p_\varepsilon}) = in(r(\varepsilon))$, and for all $w \in \Sigma^*$:

$$\begin{aligned} wt(r', w) &= wt'(\overrightarrow{r'}(w)) = wt'(r'(w), r'(w\sigma_1), \ldots, r'(w\sigma_k)) \\ &= wt'((r(w), p_w, l_{p_w}), (r(w\sigma_1), p_{w\sigma_1}, l_{p_{w\sigma_1}}), \ldots, (r(w\sigma_k), p_{w\sigma_k}, l_{p_{w\sigma_k}})) \\ &= wt(r(w), l_{p_w}, r(w\sigma_1), \ldots, r(w\sigma_k)) = wt(r(w), t(w), r(w\sigma_1), \ldots, r(w\sigma_k)) \\ &= wt(r, w). \end{aligned}$$

Thus, we obtain

$$\begin{aligned} weight(r') &= in'(r'(\varepsilon)) \otimes \bigotimes_{w \in \Sigma^*} \phi_w(wt(r', w)) \\ &= in(r(\varepsilon)) \otimes \bigotimes_{w \in \Sigma^*} \phi_w(wt(r, w)) = weight(r). \end{aligned}$$

Now suppose that $r' \in R_{\mathcal{M}_\mathcal{A}} \setminus im(\tau)$. In other words, for every $r \in R_\mathcal{M}(t)$, $r' \neq \tau(r)$, and hence

$$\exists z \in \Sigma^*,\ r'(z) \neq (r(z), p_z, l_{p_z}) \tag{3.2}$$

From r' we define three mappings $r_0 : \Sigma^* \to Q$, $p_0 : \Sigma^* \to P$, $l_0 : \Sigma^* \to L$ by setting $r'(w) = (r_0(w), p_0(w), l_0(w))$ for every $w \in \Sigma^*$. Obviously, $r_0 \in R_\mathcal{M}(t)$ (since any mapping from Σ^* to Q is a run of $\mathcal{M}$ on t). Then, from (3.2), we get that $\exists z \in \Sigma^*$ such that $p_0(z) \neq p_z$ or $l_0(z) \neq l_{p_z}$ (otherwise it would be the case that $r' = \tau(r_0)$), and assume without loss of generality that z has minimal length. We distinguish two cases.

- $z = \varepsilon$. This implies that $p_0(\varepsilon) \neq p_\varepsilon$ or $l_0(\varepsilon) \neq l_{p_\varepsilon}$. In both cases, $in(r'(\varepsilon)) = 0$ and thus $weight(r') = 0$, since $0 \otimes a = 0$ for all $a \in S$.
- $z = v\sigma_i$. This implies that $p_z \neq p_0(z)$ or $l_{p_z} \neq l_0(z)$. In the first case, we have that $(p_v, l_{p_v}, \ldots, p_0(z), \ldots) \neq (p_v, l_{p_v}, \ldots, p_{v\sigma_i}, \ldots)$. Since $(p_v, l_{p_v}, \ldots, p_{v\sigma_i}, \ldots)$ is the unique p_v-transition, we get that $(p_v, l_{p_v}, \ldots, p_0(z), \ldots) \notin \Delta$, yielding $wt(r', v) = 0$. In the second case, $(p_z, l_0(z), \ldots) \neq (p_z, l_{p_z}, \ldots)$ and thus $(p_z, l_0(z), \ldots) \notin \Delta$, yielding $wt(r', z) = 0$. In both cases, we get that $weight(r') = 0$

Finally, since $(S, \oplus, 0)$ is a commutative monoid, we get that:

$$\begin{aligned}(\|\mathcal{M}_\mathcal{A}\|, t_{ul}) &= \bigoplus_{r' \in R_{\mathcal{M}_\mathcal{A}}} weight(r') = \bigoplus_{r' \in im(\tau)} weight(r') \\ &= \bigoplus_{r \in R_\mathcal{M}(t)} weight(\tau(r)) = \bigoplus_{r \in R_\mathcal{M}(t)} weight(r) \\ &= (\|\mathcal{M}\|, t)\end{aligned}$$

□

Thus, it remains to show how the behavior of a Φ-wLTA working on the unlabeled tree can be computed. For wLTAs (without discounting) over complete distributive lattices this was done in [BP10]. In the next section, we show how the behavior of a Φ-wLTA over the semiring $\mathbb{R}_{\text{inf}}$ can be computed.

3.7 Computing the behavior on the unlabeled tree in $\mathbb{R}_{\text{inf}}$

Concentrating on $\mathbb{R}_{\text{inf}}$ is motivated, on the one hand, by the fact that our motivating examples (the language distances d_0, d_1 and d_2) can be expressed using wLTA with discounting over this semiring. On the other hand, discounting for this semiring is well-understood [DK06] and nicely behaved. Note, however, that our algorithms can be extended to the Viterbi semiring. Still, in order to get analogous complexity results, further computability and/or precision considerations have to be taken into account (see comments at the end of each section).

Recall that, for $\mathbb{R}_{\text{inf}}$, all endomorphisms are of the form $\overline{p}(a) = p \cdot a$ for $p \in \mathbb{R}_{\geq 0}$, and thus the discounting is of the form $\Phi = (\overline{p_1}, \ldots, \overline{p_k})$. Given $w = \sigma_{i_1} \ldots \sigma_{i_m} \in \Sigma^*$, we set $p_w = p_{i_1} \cdot \ldots \cdot p_{i_m}$ where the empty product (case $w = \varepsilon$) is 1. Then

$$\phi_w(a) = \phi_{i_1} \circ \cdots \circ \phi_{i_m}(a) = p_{i_1} \cdot \ldots \cdot p_{i_m} \cdot a = \overline{p_w}(a),$$

and thus $\phi_w = \overline{p_w}$. It is easy to see that, for $p > 0$, $\overline{p}$ distributes over inf and $\sum$. In the following, we assume that $p_i \neq 0$ for $i = 1, \ldots, k$, and we will write $p_w \cdot a$ instead of $\phi_w(a)$.

A *q-run* r of $\mathcal{M}$ is a run with $r(\varepsilon) = q$. By a slight abuse of notation, in this section we use $R(q)$ to denote the set of all q-runs of $\mathcal{M}$. The *running weight* of a q-run is defined like its weight, but without taking the initial distribution into account, i.e., $rweight(r) := \sum_{w \in \Sigma^*} p_w \cdot wt(r, w)$, and thus $weight(r) = in(q) + rweight(r)$. Consequently, if we define

$$\mu(q) := \inf_{r \in R(q)} rweight(r) \quad \text{(for every } q \in Q\text{)}$$

then $(\|\mathcal{M}\|, t_{ul}) = \min_{q \in Q} \{in(q) + \mu(q)\}$. Hence, in order to compute the behavior of $\mathcal{M}$ on t_{ul}, it suffices to calculate the values $\mu(q)$ for all $q \in Q$.

The following lemma provides recursive equations that are useful to achieve this goal.

Lemma 3.32. *For every state $q \in Q$ it holds that*

$$\mu(q) = \min_{(q_1, \ldots, q_k) \in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{i=1}^{k} p_i \cdot \mu(q_i) \right\}.$$

Proof. With simple (but tedious) application of the definitions we obtain

$$\begin{aligned}
\mu(q) &= \inf_{r \in R(q)} rweight(r) = \inf_{r \in R(q)} \sum_{w \in \Sigma^*} p_w \cdot wt(r, w) \\
&= \inf_{r \in R(q)} \left\{ p_\varepsilon \cdot wt(r, \varepsilon) + \sum_{w \in \Sigma^+} p_w \cdot wt(r, w) \right\} \\
&= \inf_{r \in R(q)} \left\{ wt(r, \varepsilon) + \sum_{w \in \Sigma^+} p_w \cdot wt(r, w) \right\} \\
&= \inf_{r \in R(q)} \left\{ wt(r, \varepsilon) + \sum_{w \in \Sigma^*} p_{\sigma_1 w} \cdot wt(r, \sigma_1 w) + \cdots + \sum_{w \in \Sigma^*} p_{\sigma_k w} \cdot wt(r, \sigma_k w) \right\} \\
&= \inf_{r \in R(q)} \left\{ wt(r, \varepsilon) + \sum_{i=1}^{k} \sum_{w \in \Sigma^*} p_{\sigma_i w} \cdot wt(r, \sigma_i w) \right\} \\
&= \min_{(q_1, \ldots, q_k) \in Q^k} \inf_{\substack{(r_1, \ldots, r_k) \in \\ R(q_1) \times \cdots \times R(q_k)}} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{i=1}^{k} p_i \cdot \sum_{w \in \Sigma^*} p_w \cdot wt(r_i, w) \right\} \\
&= \min_{(q_1, \ldots, q_k) \in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \inf_{\substack{(r_1, \ldots, r_k) \in \\ R(q_1) \times \cdots \times R(q_k)}} \sum_{i=1}^{k} p_i \cdot \sum_{w \in \Sigma^*} p_w \cdot wt(r_i, w) \right\} \\
&= \min_{(q_1, \ldots, q_k) \in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{i=1}^{k} \inf_{r \in R(q_i)} p_i \cdot \sum_{w \in \Sigma^*} p_w \cdot wt(r, w) \right\} \\
&= \min_{(q_1, \ldots, q_k) \in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{i=1}^{k} p_i \cdot \inf_{r \in R(q_i)} \sum_{w \in \Sigma^*} p_w \cdot wt(r, w) \right\} \\
&= \min_{(q_1, \ldots, q_k) \in Q^k} \left\{ wt(q, q_1, \ldots, q_k) + \sum_{i=1}^{k} p_i \cdot \mu(q_i) \right\}
\end{aligned}$$

□

Note that the exact same computations can be made for the Viterbi semiring $([0,1], \sup, \cdot, 0, 1)$, with the difference that $\inf, \min, +, \sum, \infty, 0$ are replaced by $\sup, \max, \cdot, \prod, 0, 1$ respectively, and that endomorphisms are of the form $\tilde{p}(a) = a^p$. Thus, we would obtain equations of the form

$$\mu(q) = \max_{(q_1,\ldots,q_k)\in Q^k} \left\{ wt(q, q_1, \ldots, q_k) \cdot \prod_{j=1}^{k} \mu(q_i)^{p_j} \right\}. \tag{3.3}$$

Our approach for computing the values $\mu(q)$ depends on the kind of discounting used.

3.7.1 Behavior for nondecreasing discounting

In this section we assume that the discounting is *nondecreasing*, i.e., $p_i \geq 1$ for all $i = 1, \ldots, k$. Note that absence of discounting corresponds to the special case where $p_i = 1$ for all $i = 1, \ldots, k$.

If the discounting is nondecreasing, then we have for every run $r \in R_{\mathcal{M}}$ that

$$rweight(r) = \sum_{w\in\Sigma^*} p_w \cdot wt(r, w) \geq \sum_{w\in\Sigma^*} wt(r, w),$$

where in the latter infinite sum only finitely many distinct non-negative real numbers occur. Consequently, this sum (and thus the original sum as well) is a finite number iff only 0 is used infinitely often in the sum. Therefore, a run r has finite weight iff, from a certain depth on, it has only zero-weight transitions. Consequently, we can restrict our attention to deciding for each state q whether such a (finite weight) q-run exists, and compute the smallest weight among all of them.

The first step consists of computing the set of states in Q that admit a run with only zero-weight transitions. Clearly, these are exactly the states q for which $\mu(q) = 0$. By keeping only transitions with weight 0 and then applying the emptiness test for LTAs [BT01] to the resulting automaton, these states can easily be computed.

More precisely, the computation can be done as follows. Let $\Delta_0 \subseteq Q^{k+1}$ be the set containing only the transitions in $\mathcal{M}$ with zero weight:

$$\Delta_0 := \{(q, q_1, \ldots, q_k) \in Q^{k+1} \mid wt(q, q_1, \ldots, q_k) = 0\}$$

and B_0 the subset of Q containing all the states that have no transition in Δ_0, i.e.,

$$B_0 := \{q \in Q \mid \forall (q_1, \ldots, q_k) \in Q^k.\ (q, q_1, \ldots, q_k) \notin \Delta_0\}$$

Then, we define the following iteration for $i \geq 0$:

$$B_{i+1} := B_i \cup \{q \in Q \mid \forall (q, q_1, \ldots, q_k) \in \Delta_0.\ \exists i.\ q_i \in B_i\}$$

The iteration becomes stable after at most $\ell \leq |Q|$ steps. The set $Q_\ell = Q \setminus B_\ell$ is then the set of states that admit a run with only zero-weight transitions, as the following lemma shows.

Lemma 3.33. *$q \in Q_\ell \iff \exists r \in R(q)$ that has only transitions with weight 0.*

Proof. Let $q \in Q_\ell$. This means that there is a transition $(q, q_1, \ldots, q_k) \in \Delta_0$ such that $(q_1, \ldots, q_k) \in (Q_\ell)^k$. Iterating this argument for the successor states one can build the wanted run r.

For the opposite direction, suppose that $q \notin Q_\ell$ and thus $q \in B_\ell$. Assume that j is the least index such that $q \in B_j$. By induction on j we will prove that there is no q-run that has only zero-transitions, i.e., transitions from Δ_0. If $j = 0$, i.e., $q \in B_0$, then there is no zero-weight transition starting from q, and thus no q-run with only zero-weight transitions. If $j > 0$, this implies that for every $(q, q_1, \ldots, q_k) \in \Delta_0$ exists some i such that $q_i \in B_{j-1}$. By the induction hypothesis, there is no q_i-run with only zero-weight transitions, and thus the same holds for q. □

The following lemma is a straightforward consequence of the previous one and the definition of $\mu(q)$.

Lemma 3.34. $q \in Q_\ell \iff \mu(q) = 0$

Proof. Suppose that $q \in Q_\ell$. Then $\exists r \in R(q)$ such that $rweight(r) = 0$, and thus $\mu(q) = 0$. Conversely, $q \notin Q_\ell$ implies that $\forall r \in R(q)$ there is a transition with non-zero weight. Let a_0 be the least non-zero weight among all transitions in the automaton. Then, $\forall r \in R(q)$, $\sum_{w \in \Sigma^*} wt(r, w) \geq a_0$, and thus $\mu(q) \geq a_0 > 0$. □

Summing up, the above construction gives us the next lemma.

Lemma 3.35. *The set of states* $Q_{\mu=0} := \{q \in Q \mid \mu(q) = 0\}$ *can be computed in polynomial time.*

A run with finite weight does not use a transition with weight $+\infty$ and below a certain depth in the tree it contains only states that belong to $Q_{\mu=0}$. Thus, the states used in the run must have access to states in $Q_{\mu=0}$ through transitions with finite weight. To be more precise, define the set Q_{acc} of states that *have access to* $Q_{\mu=0}$ to be the least subset of Q such that (i) $Q_{\mu=0} \subseteq Q_{acc}$ and (ii) if $q_i \in Q_{acc}$ for every $i = 1, \ldots, k$ and $wt(q, q_1, \ldots, q_k) \neq +\infty$ then $q \in Q_{acc}$. States q that have access to $Q_{\mu=0}$ have a q-run with finite running weight, and hence $\mu(q) < +\infty$. If q does not have access to $Q_{\mu=0}$, then $\mu(q) = +\infty$. By using an approach inspired by Dijkstra's shortest path algorithm, we can compute the states that have access to $Q_{\mu=0}$ together with their μ-value in polynomial time.

Initially, note that in case $Q_\ell = \emptyset$, no state has access to Q_ℓ and thus $\mu(q) = \infty$ for all $q \in Q$.

To compute $\mu(q)$ for all the states q we use the following algorithm. Set $S_0 = Q_\ell$ and consider the function

$$m^0(q) = \begin{cases} 0, & \text{if } q \in S_0 \\ \min\limits_{(q_1, \ldots, q_k) \in (S_0)^k} wt(q, q_1, \ldots, q_k), & \text{otherwise} \end{cases}$$

Next, for $i > 0$, iteratively do the following:

- $S_i := S_{i-1} \cup \{s_i\}$, for some $s_i = \arg\min_{q \notin S_{i-1}} m^{i-1}(q)$.

- For all $q \notin S_i$, update their m value:

$$m^i(q) := \min\left\{ m^{i-1}(q), \min_{(q_1,\ldots,q_k)\in(S_i)^k}\left\{ wt(q,q_1,\ldots,q_k) + \sum_{j=1}^{k} p_j \cdot m^{i-1}(q_j)\right\}\right\}$$

  while for all $q \in S_i$, $m^i(q) := m^{i-1}(q)$.

This iteration terminates after $f = |Q \setminus Q_\ell|$ steps. Moreover, since by Lemma 3.32 we know that $\mu(q)$ corresponds to:

$$\mu(q) = \min_{(q_1,\ldots,q_k)\in Q^k}\left\{ wt(q,q_1,\ldots,q_k) + \sum_{j=1}^{k} p_j \cdot \mu(q_j)\right\},$$

based on the definition of m^i and the fact that $(S_i)^k \subseteq Q^k$ it can be shown by induction on i that:

$$\mu(q) \leq m^i(q), \text{ for all } q \in Q \text{ and } i \geq 0 \tag{3.4}$$

We now show that, upon termination, $m^f(q) = \mu(q) = \inf_{r\in R(q)} rweight(r)$ holds for all $q \in Q$. From this, the behavior of $\mathcal{M}$ can be directly computed, since it can then be expressed as:

$$\|\mathcal{M}\| = \min_{q\in Q}\big(in(q) + m^f(q)\big)$$

Lemma 3.36. *For all $i \geq 0$ and $s \in S_i$, it holds that:*

1. *$\mu(s) \leq \mu(q)$ for all $q \notin S_i$, and*
2. *$\mu(s) = m^i(s)$.*

Proof. We prove our claims by induction on i.

Base case. $i = 0$. Since $s \in S_0$ and $S_0 = Q_\ell$, by Lemma 3.34 and the definition of m^0, it follows that $\mu(s) = m^0(s) = 0$. In addition, let $q \notin S_0$ be a state in Q. Since $\mu(q) \in (\mathbb{R}_{\geq 0} \cup \{\infty\})$, this means that $\mu(s) \leq \mu(q)$.

Induction step. We show our claims hold for all $i > 0$, based on the assumption that they hold for all numbers smaller than i.

Let $S_i = S_{i-1} \cup \{s_i\}$. The application of induction yields $\mu(s) = m^{i-1}(s)$ and $\mu(s) \leq \mu(s_i)$ for all $s \in S_{i-1}$. Since $m^i(s) = m^{i-1}(s)$ for all $s \in S_{i-1}$, this means that $\mu(s) = m^i(s)$. Hence, it remains to show that the claims hold for s_i.

1. We want to show that $\mu(s_i) \leq \mu(q)$ for all $q \notin S_i$. Suppose for a contradiction that there exists $s' \notin S_i$ such that $\mu(s') < \mu(s_i)$. Without loss of generality, s' is selected such that $\mu(s') \leq \mu(q)$ for all $q \notin S_i$. Based on Lemma 3.32, $\mu(s')$ can be expressed as:

$$\mu(s') = wt(s', q_1^0, \ldots, q_k^0) + \sum_{j=1}^{k} p_j \cdot \mu(q_j^0), \tag{3.5}$$

   for some tuple $(q_1^0, \ldots, q_k^0) \in Q^k$. Let us first show that $(q_1^0, \ldots, q_k^0) \in (S_{i-1})^k$. Suppose on the contrary that $q_j^0 \notin S_{i-1}$ for some $1 \leq j \leq k$. Then, it clearly holds that $0 < \mu(s') \leq \mu(q_j^0)$ by the way s' was selected. We distinguish the following two cases:

- $wt(s', q_1^0, \ldots, q_k^0) > 0$.Consequently, $\mu(s') > \mu(q_j^0)$ must hold[8], a contradiction.
- $wt(s', q_1^0, \ldots, q_k^0) = 0$. To avoid the same contradiction as before, it must be the case that $\mu(q_\lambda^0) = 0$ for all $\lambda \neq j$. Hence, $\mu(s') = \mu(q_j^0)$, and q_j^0 could have been chosen as s'. By iterating this argument, while assuming the corresponding transition has zero-weight, we will end up with s' having a run with only zero-weight transitions, i.e., belonging to $Q_{\mu=0}$[9]. Thus, we obtain a contradiction since $s' \notin S_0 = Q_\ell = Q_{\mu=0}$.

Then, having $(q_1^0, \ldots, q_k^0) \in (S_{i-1})^k$ implies that $\mu(q_j^0) = m^{i-1}(q_j^0)$ for all $1 \leq j \leq k$ (by induction). Recall that $m^i(s')$ corresponds to the expression:

$$m^i(s') = \min\left\{ m^{i-1}(s'), \min_{(q_1,\ldots,q_k)\in(S_i)^k} \left\{ wt(s', q_1, \ldots, q_k) + \sum_{j=1}^{k} p_j \cdot m^{i-1}(q_j) \right\} \right\}$$

Since $(S_{i-1})^k \subseteq (S_i)^k$, this means that the value corresponding to the inner minimization in the previous expression is not greater than $\mu(s')$. Therefore, using (3.4) we obtain:

$$m^i(s') = \mu(s') < \mu(s_i) \leq m^i(s_i) = m^{i-1}(s_i) \tag{3.6}$$

Moreover, if $m^i(s') < m^{i-1}(s')$ holds, this must have been a direct consequence of introducing s_i to form S_i. In other words, $m^i(s')$ is updated by using the value $m^{i-1}(s_i)$, which means that $m^{i-1}(s_i) \leq m^i(s')$. The latter is obviously not consistent with (3.6). Therefore, in order to still be consistent with $\mu(s') < \mu(s_i)$, the equality $m^i(s') = m^{i-1}(s')$ must hold. But then, it follows that $m^{i-1}(s') < m^{i-1}(s_i)$ which is a contradiction since s_i was selected to obtain S_i. Thus, we can conclude that $\mu(s_i) \leq \mu(q)$ for all $q \notin S_i$.

2. Consider $\mu(s_i)$ expressed as in (3.5) (substitute s' by s_i). Since we have just shown that $\mu(s_i) \leq \mu(q)$ for all $q \notin S_i$, similarly as before, it can be proved that $(q_1^0, \ldots, q_k^0) \in (S_{i-1})^k$ holds for s_i as well. Then, the same argument used in 1. yields that $m^i(s_i) \leq \mu(s_i)$. Thus, $m^i(s_i) = \mu(s_i)$ holds (using (3.4)). □

Summing up, computing Q_ℓ requires quadratic time and computing $m^f(q)$ for every $q \in Q$ requires cubic time. Since the behavior of $\mathcal{M}$ on t_{ul} can easily be computed from the values $\mu(q)$ for $q \in Q$, this yields the following theorem.

Theorem 3.37. *The behavior of a Φ-wLTA with nondecreasing discounting Φ over $\mathbb{R}_{\inf}$ on the unlabeled tree can be computed in polynomial time.*

Note that a similar algorithm can be utilized for the behavior over the Viterbi semiring. As was done for obtaining Equation (3.3), replace $\inf, \min, +, \sum, \infty, 0$ by $\sup, \max, \cdot, \prod, 0, 1$ respectively. Also recall that endomorphisms are of the form $\tilde{p}(a) = a^p$. Other than this, the algorithm and the proof of its correctness is the same. However, since we might have to compute numbers as small as $x^{p^{|Q|}}$, the algorithm might take exponential time because of their computation.

[8] Recall that $p_j \geq 1$.

[9] This iteration terminates in at most $|\, Q \setminus Q_\ell \,|$ steps.

3.7.2 Behavior for contracting discounting

In this section, we assume a *contracting* discounting, i.e., $p = \sum_{i=1}^{k} p_i < 1$.

Recall that it suffices to compute the value $\mu(q)$ for every $q \in Q$. Let $Q = \{q_1, \ldots, q_n\}$. For each $q_i \in Q$, the unknown value $\mu(q_i)$ is associated to a variable x_i. Additionally, let $I = \{1, \ldots, n\}$. Then, Lemma 3.32 states that $(\mu(q_1), \ldots, \mu(q_n))$ is a solution of the following system of equations:

$$x_i = \min_{(i_1,\ldots,i_k)\in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot x_{i_j} \right\} \tag{3.7}$$

In the following, we will make use of notions and results from metric topology. Refer to the Appendix for the relevant definitions.

For all $q_i \in Q$, we define a function $f_i : \mathbb{R}^n \to \mathbb{R}$ as follows:

$$f_i(a_1, \ldots, a_n) = \min_{(i_1,\ldots,i_k)\in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot a_{i_j} \right\}$$

Next, we define the vector function $f : \mathbb{R}^n \to \mathbb{R}^n$ as:

$$f(a_1, \ldots, a_n) = (f_1(a_1, \ldots, a_n), \ldots, f_n(a_1, \ldots, a_n))$$

Clearly, a vector $\mathbf{a} = (a_1, \ldots, a_n)$ is a solution of the system of equations in (3.7) iff it is a fixed point of f, i.e., $f(\mathbf{a}) = \mathbf{a}$. Thus, it is enough to show that f is indeed a contraction on a complete metric space $(\mathbb{R}^n, d)$.

Lemma 3.38. *The function f defined above is a contraction on $(\mathbb{R}^n, d_\infty)$.*

Proof. Given $\mathbf{a} = (a_1, \ldots, a_n), \mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{R}^n$, recall that $d_\infty(\mathbf{a}, \mathbf{b}) = \max_{i\in I} |a_i - b_i|$.

For every $i \in I$ we have:

$$\begin{aligned} f_i(\mathbf{a}) &= \min_{(i_1,\ldots,i_k)\in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot a_{i_j} \right\} \\ &= wt(q_i, q_{i_1^0}, \ldots, q_{i_k^0}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^0} \end{aligned}$$

for a particular $(i_1^0, \ldots, i_k^0) \in I^k$, and likewise

$$f_i(\mathbf{b}) = wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot b_{i_j^1}.$$

Since for $\mathbf{a}$, the minimum is achieved for $(i_1^0, \ldots, i_k^0)$, it holds that:

$$wt(q_i, q_{i_1^0}, \ldots, q_{i_k^0}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^0} \leq wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^1}.$$

Without loss of generality, assume that $f_i(\mathbf{b}) \leq f_i(\mathbf{a})$. Thus, we have:

$$\begin{aligned}
|f_i(\mathbf{a}) - f_i(\mathbf{b})| &= f_i(\mathbf{a}) - f_i(\mathbf{b}) \\
&= wt(q_i, q_{i_1^0}, \ldots, q_{i_k^0}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^0} - \left(wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot b_{i_j^1} \right) \\
&\leq wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot a_{i_j^1} - \left(wt(q_i, q_{i_1^1}, \ldots, q_{i_k^1}) + \sum_{j=1}^{k} p_j \cdot b_{i_j^1} \right) \\
&= \sum_{j=1}^{k} p_j \cdot (a_{i_j^1} - b_{i_j^1}) \leq \sum_{j=1}^{k} p_j \cdot \max_{i \in I} |a_i - b_i| \\
&= p \cdot \max_{i \in I} |a_i - b_i| = p \cdot d_\infty(\mathbf{a}, \mathbf{b})
\end{aligned}$$

Overall, we get that $|f_i(\mathbf{a}) - f_i(\mathbf{b})| \leq p \cdot \ d_\infty(\mathbf{a}, \mathbf{b})$ for every $i \in I$, and thus

$$d_\infty(f(\mathbf{a}), f(\mathbf{b})) = \max_{i \in I} |f_i(\mathbf{a}) - f_i(\mathbf{b})| \leq p \cdot \ d_\infty(\mathbf{a}, \mathbf{b}) < d_\infty(\mathbf{a}, \mathbf{b}),$$

where the last inequality holds since $p < 1$, and we have that f is a contraction. □

From Banach's fixed point theorem, we have that f has a unique fixed point, and thus the system of equations (3.7) has a unique solution. Thus, to compute the values $\mu(q)$ for $q \in Q$ it is sufficient to compute this unique solution. This can be realized using Linear Programming [Sch99].

Definition 3.39. *A* Linear Programming problem *or* LP problem *is a set of restrictions along with an objective function. In its most general form, an LP problem looks like this:*

$$\begin{aligned}
\textit{objective}: \quad & \textit{min/max } z = c_1 x_1 + \ldots + c_n x_n \\
\textit{restrictions}: \quad & a_{1,1} x_1 + \ldots + a_{1,n} x_n \lessgtr b_1 \\
& \vdots \\
& a_{m,1} x_1 + \ldots + a_{m,n} x_n \lessgtr b_m
\end{aligned}$$

where $a_{i,j}, b_i, c_j$ *are rational numbers.*

The feasible region *of the LP problem consists of all the tuples* $(x_1, \ldots, x_n)$ *that satisfy the restrictions. The answer to an LP problem is a tuple in the feasible region that maximizes the objective function and "no" if the feasible region is empty.* ♢

It is well known that LP problems are solvable in polynomial time in the size of the problem [Sch99].

From the above system of equations (3.7) we can derive an LP problem. Consider for every $i \in I$, $(i_1, \ldots, i_k) \in I^k$ the inequation

$$x_i \leq wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot x_{i_j} \tag{3.8}$$

and the objective

$$z = \max \sum_{i \in I} x_i. \tag{3.9}$$

Lemma 3.40. *The LP problem consisting of the inequations* (3.8)*, and the objective* (3.9) *has the unique solution*

$$\{x_i \mapsto \mu(q_i) \mid i \in I\}.$$

Proof. Initially, observe that the above vector is in the feasible region, since it satisfies the restrictions (3.8). Next, we procede to show that it is indeed the only point that maximizes the objective function. First, we need the following claim.

Claim 3.41. *If* **a** *is a solution that maximizes the objective function then, for every $i \in I$, at least one of the inequalities* (3.8) *holds as an equality.*

Claim. Suppose on the contrary that **c** is a solution that maximizes z, but for some $i \in I$, inequalities $c_i \leq wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot c_{i_j}$ are strict for all $(i_1, \ldots, i_k) \in I^k$. This would mean that the value of c_i can be increased, and all inequalities would still hold; the increase in c_i might increase the right-hand side of some other inequality, but since the left-hand side remains the same, all restrictions are satisfied. Thus, a new point $\mathbf{c}'$ has been produced that satisfies all the restrictions of the LP problem and additionally gives a larger value for the objective function. This is a contradiction to our initial assertion about **c**. This completes the proof of the claim.

As a result, any points that are solutions to the LP problem, satisfy the condition

$$x_i = \min_{(i_1,\ldots,i_k) \in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) + \sum_{j=1}^{k} p_j \cdot x_{i_j} \right\}$$

for all $i \in I$. Thus, they correspond to solutions of the system of equations (3.7).

Finally, since there is a unique such solution, the solution of the LP problem is this unique solution, the vector $(\mu(q_1), \ldots, \mu(q_n))$. □

Since solutions of Linear Programming problems can be computed in polynomial time, this is also the case for the values $\mu(q)$, and thus for the behavior.

Theorem 3.42. *The behavior of a Φ-wLTA with contracting discounting Φ over $\mathbb{R}_{\inf}$ on the unlabeled tree can be computed in polynomial time.*

Note that for the Viterbi semiring we get analogous equations:

$$y_i = \max_{(i_1,\ldots,i_k) \in I^k} \left\{ wt(q_i, q_{i_1}, \ldots, q_{i_k}) \cdot \prod_{j=1}^{k} {y_{i_j}}^{p_j} \right\}$$

By taking the logarithm of these equations, we derive equations of the form (3.7). However, the numbers might no longer be rational, and thus further computational issues should be taken into consideration.

3.8 More results on tree automata

Before we conclude this chapter, we will provide some more technical results on LTAs that will prove useful later in this thesis.

As we have already seen in Definition 3.22, we can use trees with an arbitrary label set to define languages. Furthermore, as we discussed in the Introduction, there is a correspondence between *finite* languages and $\mathcal{FL}_0$ concept descriptions. In Chapters 4 and 6 we will use the runs of an LTA, which are essentially trees, to define languages, that we then want to use to build $\mathcal{FL}_0$ concept descriptions. It is hence essential to be able to figure out whether any of these runs corresponds to a finite language. This is indeed possible, as the following result states.

Lemma 3.43. *Let $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$ be an LTA of size n. Given a set of states $B \subseteq Q$, we can check whether $\mathcal{A}$ has a run that uses states from B only finitely many times in time $h(2^{|\Sigma|}n)$ for a polynomial h.*

In order to perform this check, we can adapt an approach from [BO12] that uses a special kind of Büchi tree automata. The main idea is to transform the LTA into a BTA that picks a point (word) in every path after which it will not use states from B any more. This guarantees that the states from B occur only finitely often in the entire tree.

Initially, we create a copy of non-B states. At any point the old automaton could reach a non-B state, the new automaton has the option to go to the copy of this state, while copied states can only reach copied (and hence non-B) states and no original states.

More precisely, given an LTA $\mathcal{A} = (\Sigma, Q, L, \Delta, I)$ and $B \subseteq Q$, construct the Büchi tree automaton $\mathcal{B} = (\Sigma, Q', L, \Delta', I', F)$ as follows:

$$
\begin{array}{lcl}
Q' & = & Q \cup \{q' \mid q \in Q \setminus B\}, \\
I' & = & I \cup \{q' \mid q \in I \setminus B\}, \\
F & = & \{q' \mid Q \setminus B\}, \text{ and} \\
\Delta' & = & \{(p, \ell, p_1, \ldots, p_k) \in Q' \times L \times {Q'}^k \mid (1) \text{ or } (2)\}, \text{ where} \\
 & & (1)\ (p, \ell, q_1, \ldots, q_k) \in \Delta \text{ and for every } i = 1, \ldots, k,\ p_i = q_i \text{ if } q_i \in B \\
 & & \qquad\qquad \text{or } p_i \in \{q_i, q_i'\} \text{ if } q_i \notin B \\
 & & (2)(q, \ell, q_1, \ldots, q_m) \in \Delta,\ q, q_1, \ldots, q_k \notin B, \text{ and } p = q', p_1 = q_1', \ldots, p_k = q_k'.
\end{array}
$$

This BTA can be now used to perform the aforementioned check.

Lemma 3.44. *Every successful run of $\mathcal{B}$ induces a run of $\mathcal{A}$ that uses states from B only finitely many times, and conversely, every such run of $\mathcal{A}$ is induced by some successful run of $\mathcal{B}$.*

Proof. Let r be a successful run of $\mathcal{B}$ on a tree $t \in T^{\omega}_{\Sigma,L}$. Then by the definition of a successful run and the construction of $\mathcal{B}$, in every path states from $\mathcal{A}$, i.e., states in Q, appear finitely often. Furthermore, by construction of $\mathcal{B}$, once a copied state, i.e., a state in $Q' \setminus Q$ is reached in a path, no original state can be accessed after that point. From this, we infer that the original states form a tree; in particular a connected graph. Hence, if there exist infinitely many of them, by König's Lemma [Kle02], there exists an infinite path containing only original states. This is a contradiction since the run was supposed to be successful. Therefore, states from B occur finitely often. Now define the run r' of $\mathcal{A}$ by projecting all copied states of r to their original counterparts. Clearly, r' is a run of $\mathcal{A}$ on t.

The reverse procedure proves the second statement of the lemma. More precisely, consider a run r of $\mathcal{A}$ on a tree $t \in T^{\omega}_{\Sigma,L}$ that only uses states from B finitely many times. Define the run r' of $\mathcal{B}$ by replacing in every path all states after the last occurence of a state from B with their copied counterparts. By construction, this is indeed a successful run of $\mathcal{B}$. □

Hence, since the emptiness check for BTAs can be performed in polynomial time, and the size of $\mathcal{B}$ is at most $2^{|\Sigma|}|\mathcal{A}|$, we have proved Lemma 3.43.

3.9 Outlook

The main technical result of this chapter is the polynomial time procedure we have devised in order to compute the behavior of Φ-wLTAs with nondecreasing or contracting discounting over $\mathbb{R}_{\text{inf}}$ on regular trees represented by rLTAs. To the best of our knowledge, the only other work that deals with computing the behavior of weighted automata working on infinite trees is [BP10], where the authors consider weights over distributive lattices. Another method for computing the behavior of weighted automata in the same setting can be implicitly obtained from the results of [DKR08]. It would be interesting to build on this work and investigate the remaining kinds of discounting over $\mathbb{R}_{\text{inf}}$, and also computability for other semirings or weight structures.

Combining this technical result with language distances, we obtain the following result, which will be especially useful in Chapter 5.

Theorem 3.45. *Given an rLTA representing the pair of languages (K, L) and a Φ-wLTA with nondecreasing or contracting discounting over $\mathbb{R}_{\text{inf}}$ corresponding to the language distance d, the value $d(K, L)$ can be computed in polynomial time. In particular, this holds for the distances d_0, d_1, and d_2.*

Chapter 4

Unification in the Description Logic $\mathcal{FL}_0$

In this chapter we will introduce the main problem that we will try to extend in the rest of this thesis, that is unification in $\mathcal{FL}_0$, and we describe how this problem has been tackled in the classical case. After we give some basic definitions from Description Logic, we will formally define unification in this setting. We will then describe the reduction of this problem to solving language equations of a certain kind, and we will conclude this chapter by discussing how solvability of equations of that form can be checked.

4.1 The Description Logic $\mathcal{FL}_0$

We first provide the syntax and semantics of the Description Logic $\mathcal{FL}_0$, and we describe TBoxes, the structures that model background terminological knowledge. We proceed to discuss concept subsumption and how formal languages can be utilized to this end. The presentation in this section is rather succinct. A more thorough look into DLs can be found in [BCM+03].

Syntax Concept descriptions of the DL $\mathcal{FL}_0$ are built from disjoint sets of concept and role names using the concept constructors *conjunction* ($\sqcap$), *value restriction* ($\forall r.C$) and the *top concept* ($\top$). More formally, the set $\mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R})$ of $\mathcal{FL}_0$ *concept descriptions over the set of concept names* $\mathsf{N_C}$ *and the set of role names* $\mathsf{N_R}$ is obtained using the following syntax rules:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C, \tag{4.1}$$

where $A \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, and $C \in \mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R})$.

If there is no confusion, we will usually omit the sets of concept and role names (also from the notation) and simply refer to the set of $\mathcal{FL}_0$ concept descriptions ($\mathcal{C}_{\mathcal{FL}_0}$).

When considering the complexity of certain tasks related to DLs, we commonly take the size of a concept description as the input size for this task. Given an $\mathcal{FL}_0$ concept description C, its size $|C|$ is inductively defined as

$$|C| := \begin{cases} 1 & \text{if } C \in \mathsf{N_C} \cup \{\top\} \\ 1 + |D| & \text{if } C = \forall r.D \\ |D_1| + |D_2| & \text{if } C = D_1 \sqcap D_2 \end{cases}$$

At this point a clarification is needed, since there is some ambiguity in the literature concerning the sets $\mathsf{N_C}$ and $\mathsf{N_R}$. Sometimes they are assumed to be countably infinite sets, while in other occasions they are supposed to be finite, and often even fixed. For practical

applications finite sets suffice. However, regarding computational complexity, many hardness results require arbitrarily large sets. For this reason, for the problems we will define in the rest of the thesis we always assume (sometimes implicitly) that $\mathsf{N_C}$ and $\mathsf{N_R}$ are part of the input. Often, however, these sets simply contain the concept and role names occurring in the concept descriptions under consideration, and hence the size of the former is linearly bounded by the size of the latter.

Semantics The semantics of $\mathcal{FL}_0$ is defined using the notion of an *interpretation* as in first-order logic.

Definition 4.1. *An* interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ *consists of a nonempty set* $\Delta^\mathcal{I}$*, called the* interpretation domain*, and a mapping* $\cdot^\mathcal{I}$*, called the* interpretation function *that maps every concept name* $A \in \mathsf{N_C}$ *to a set* $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$*, and every role name* $r \in \mathsf{N_R}$ *to a binary relation* $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$.

The interpretation function is inductively extended to arbitrary $\mathcal{FL}_0$ *concept descriptions as follows:*

- $\top^\mathcal{I} := \Delta^\mathcal{I}$,
- $(C \sqcap D)^\mathcal{I} := C^\mathcal{I} \cap D^\mathcal{I}$,
- $(\forall r.C)^\mathcal{I} := \{d \in \Delta^\mathcal{I} \mid \forall e.((d,e) \in r^\mathcal{I} \implies e \in C^\mathcal{I})\}$. ♢

TBoxes The TBox (terminological box) contains terminological knowledge, i.e., knowledge about how different concepts are related. The restrictions given in the TBox should be satisfied by all elements of an interpretation.

Definition 4.2. *A* general TBox *is a finite set of* general concept inclusions (GCIs)*, which are expressions of the form* $C \sqsubseteq D$*, where* C *and* D *are concept descriptions.*

More restricted forms of TBoxes consist of concept definitions *instead of GCIs. A* concept definition *is an expression of the form* $A \equiv C$*, where* C *is a concept description and* $A \in \mathsf{N_C}$*. A TBox* $\mathcal{T}$ *containing only concept definitions is called* acyclic *or* unfoldable *if the following two conditions hold:*

- *for every concept name* $A \in \mathsf{N_C}$ *there is at most one concept definition of the form* $A \equiv C \in \mathcal{T}$.
- *there is no* $\{A_1 \equiv C_1, \ldots, A_n \equiv C_n\} \subseteq \mathcal{T}$ *such that* A_{i+1} *occurs in* C_i *for* $1 \leq i < n$ *and* A_1 *occurs in* C_n.

A TBox that does not satisfy the second requirement is called a cyclic *TBox.*

Concept definitions and GCIs are called TBox axioms. ♢

TBoxes consisting of concept definitions predate general TBoxes, and were the first attempt at integrating background terminological knowledge. As will become apparent from the semantics of TBoxes, a concept definition $A \equiv C$ is equivalent to the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$. Hence, general TBoxes are indeed more *general* than the ones consisting of concept definitions, and we will not deal further with concept definitions.

An interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An interpretation that satisfies each GCI in a TBox $\mathcal{T}$ is called a *model* of $\mathcal{T}$. The size of a GCI $g : C \sqsubseteq D$ is defined to be $|g| := |C| + |D|$, and the size of a TBox $\mathcal{T}$ is the sum of the sizes of the GCIs it contains, i.e., $|\mathcal{T}| := \sum_{g \in \mathcal{T}} |g|$.

Subsumption and formal languages Given a TBox $\mathcal{T}$ and concept descriptions C, D, we say that C *is subsumed by* D *with respect to* $\mathcal{T}$ (denoted as $C \sqsubseteq_{\mathcal{T}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$, that is, if in every model of $\mathcal{T}$, the concept description C is interpreted as a subset of the interpretation of D. Two concept descriptions are *equivalent with respect to* $\mathcal{T}$ (written $C \equiv_{\mathcal{T}} D$) if $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$, that is, if they are interpreted as the same set in every model $\mathcal{I}$ of $\mathcal{T}$. If $\mathcal{T}$ is the empty TBox, we simply write $C \sqsubseteq D$ and $C \equiv D$ instead of $C \sqsubseteq_{\emptyset} D$ and $C \equiv_{\emptyset} D$.

As an easy example, with respect to the empty TBox (and hence also w.r.t. any TBox) we have that $A \sqcap B \sqsubseteq A$, since for every interpretation $\mathcal{I}$ we have that $(A \sqcap B)^{\mathcal{I}} = A^{\mathcal{I}} \cap B^{\mathcal{I}} \subseteq A^{\mathcal{I}}$. Furthermore, if we consider the TBox $\mathcal{T} = \{A \sqsubseteq B\}$, for every interpretation $\mathcal{I}$ that is a model of $\mathcal{T}$ we have that $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$. This implies that $(A \sqcap B)^{\mathcal{I}} = A^{\mathcal{I}} \cap B^{\mathcal{I}} = A^{\mathcal{I}}$, and hence $A \sqcap B \equiv_{\mathcal{T}} A$.

The subsumption problem for $\mathcal{FL}_0$ is ExpTime-complete: the complexity upper bound is inherited from the more expressive DL $\mathcal{ALC}$ [Sch91], and the lower bound was shown in [BBL05]. In [BFP18] an alternative way of proving the upper bound is provided. It is based on the characterization of subsumption that uses formal language inclusion. Traces of this idea can already be found in [BL84], where it is shown that subsumption (and thus also equivalence) for (an extention of) $\mathcal{FL}_0$ w.r.t. the empty TBox can be decided in polynomial time. We will now introduce this characterization for the empty TBox,[1] before we describe the more general result from [BFP18].[2]

This characterization relies on transforming $\mathcal{FL}_0$ concept descriptions into an appropriate normal form as follows. First, the semantics given to concept constructors in $\mathcal{FL}_0$ implies that value restrictions distribute over conjunction, i.e., for all $\mathcal{FL}_0$ concept descriptions C, D and role names r it holds that $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$. Furthermore, we have that $\top \sqcap C \equiv C$, $C \sqcap \top \equiv C$, and $\forall r.\top \equiv \top$. Using these equivalences as rewrite rules from left to right, each $\mathcal{FL}_0$ concept description can be translated into an equivalent one that is either $\top$ or a conjunction of concept descriptions of the form $\forall r_1 \ldots \forall r_n.A$, where $\{r_1, \ldots, r_n\} \subseteq \mathsf{N_R}$ and $A \in \mathsf{N_C}$. Such expressions can be abbreviated as $\forall w.A$, where w represents the word $r_1 \ldots r_n$. Note that $n = 0$ means that w is the empty word ε, and thus $\forall \varepsilon.A$ corresponds to A. Furthermore, a conjunction of the form $\forall w_1.A \sqcap \cdots \sqcap \forall w_m.A$ can be written as $\forall L.A$ where $L \subseteq \mathsf{N_R}^*$ is the finite language $\{w_1, \ldots, w_m\}$. We use the convention that $\forall \emptyset.A$ corresponds to the top concept $\top$. Thus, any two $\mathcal{FL}_0$ concept descriptions can be represented as

$$C \equiv \forall K_1.A_1 \sqcap \ldots \sqcap \forall K_k.A_k, \quad D \equiv \forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k, \tag{4.2}$$

where $A_1, \ldots, A_k$ are the concept names occurring in C and D, and $K_1, \ldots, K_k, L_1, \ldots, L_k$ are finite languages over the alphabet of role names $\mathsf{N_R}$, i.e., finite subsets of $\mathsf{N_R}^*$. In case the

[1] Since [BL84] is quite verbose, and the connection is not explicitly mentioned, we will follow the line of presentation of [BN01].

[2] The purpose of providing the special result first is not only didactic or historical. In Chapter 6, we will only be able to extend unification in the approximate setting only for the case of the empty TBox. Hence, it will come in handy to explicitly mention the relevant results.

concept name A_i occurs in C, but not in D, then $L_i = \emptyset$, and thus $\forall L_i.A_i \equiv \top$. Concept names occurring in D, but not in C, are treated analogously. For example, the concept descriptions $C = A_1 \sqcap \forall r_1.(A_1 \sqcap \forall r_1.A_2 \sqcap \forall r_2.A_1) \sqcap \forall r_2.(A_2 \sqcap A_3)$ and $D = \forall r_1.(A_1 \sqcap \forall r_2.A_1) \sqcap \forall r_2.A_2$ have the normal forms $\forall\{\varepsilon, r_1, r_1r_2\}.A_1 \sqcap \forall\{r_1r_1, r_2\}.A_2 \sqcap \forall\{r_2\}.A_3$ and $\forall\{r_1, r_1r_2\}.A_1 \sqcap \forall\{r_2\}.A_2 \sqcap \forall\emptyset.A_3$ respectively.

Using this representation, subsumption between the $\mathcal{FL}_0$ concept descriptions C, D can be characterized as follows [BN01]:

$$C \sqsubseteq D \text{ iff } L_i \subseteq K_i \text{ holds for all } i, 1 \leq i \leq k.$$

Since these normal forms can be obtained in time polynomial in the size of the original concept descriptions, checking subsumption with an empty TBox can be performed in polynomial time [BN01].

Note that in our previous example, we have that $C \sqsubseteq D$ and the corresponding inclusions between the languages in the normal forms hold.

In the presence of a non-empty TBox $\mathcal{T}$, a similar characterization of subsumption and equivalence can be obtained [Pen15; BFP18], but now the definition of the languages needs to take the GCIs in $\mathcal{T}$ into account. Next, we recall this characterization.

Definition 4.3. *Given an $\mathcal{FL}_0$ concept description C, a concept name A and an $\mathcal{FL}_0$ TBox $\mathcal{T}$, the* value restriction set *of C w.r.t. A and $\mathcal{T}$ is defined as:*

$$\mathcal{L}_\mathcal{T}(C,A) := \{w \in \mathsf{N_R}^* \mid C \sqsubseteq_\mathcal{T} \forall w.A\}.$$

If $\mathsf{N_C} = \{A_1, \ldots, A_k\}$ we call the tuple of languages $\mathcal{L}_\mathcal{T}(C) = (\mathcal{L}_\mathcal{T}(C,A_1), \ldots, \mathcal{L}_\mathcal{T}(C,A_k))$ the value restriction set *of C w.r.t. $\mathcal{T}$.* ♢

Using the value restriction sets, subsumption of $\mathcal{FL}_0$ concepts w.r.t. an $\mathcal{FL}_0$ TBox can be characterized via language inclusions as for the empty TBox case.

Proposition 4.4 ([BFP18]). *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C, D $\mathcal{FL}_0$ concept descriptions. Then $C \sqsubseteq_\mathcal{T} D$ iff $\mathcal{L}_\mathcal{T}(D,A) \subseteq \mathcal{L}_\mathcal{T}(C,A)$ for every concept name A occurring in C or D. Likewise, for equivalence we have that $C \equiv_\mathcal{T} D$ iff $\mathcal{L}_\mathcal{T}(C,A) = \mathcal{L}_\mathcal{T}(D,A)$ for every A.* ♢

Characterizing an $\mathcal{FL}_0$ concept description C with normal form $C = \forall K_1.A_1 \sqcap \cdots \sqcap \forall K_k.A_k$ by its value restriction sets $\mathcal{L}_\mathcal{T}(C,A_i)$ generalizes determining the finite languages K_i $(1 \leq i \leq k)$ in the normalization step. Indeed, for the empty TBox we have $\mathcal{L}_\emptyset(C,A_i) = K_i$ for $i = 1, \ldots k$. For a general TBox $\mathcal{T}$, the languages $\mathcal{L}_\mathcal{T}(C,A)$ may be infinite. A simple example is the $\mathcal{FL}_0$ concept description A and the TBox $\mathcal{T} = \{A \sqsubseteq \forall r.A\}$. One can easily see that $A \sqsubseteq_\mathcal{T} \forall w.A$ for all $w \in r^*$, which means that $\mathcal{L}_\mathcal{T}(C,A) = r^*$. Therefore, to determine subsumption of two concepts by comparing the respective value restriction sets, inclusion between infinite sets must be considered. Since $\mathcal{L}_\mathcal{T}(C)$ is a tuple of languages, it can be represented as a tree in the lines of Definition 3.8. For ease of notation, when there is no confusion on the TBox that is being used, we will denote this tree by t_C instead of $t_{\mathcal{L}_\mathcal{T}(C)}$.

Example 4.5. *Let C be the $\mathcal{FL}_0$ concept description $C := A \sqcap \forall s.(A \sqcap B)$ and $\mathcal{T}$ the TBox $\mathcal{T} = \{A \sqsubseteq \forall r.A, B \sqsubseteq \forall s.B\}$. With basic calculations, we obtain that the value restriction sets for A and B are*

$$\mathcal{L}_\mathcal{T}(C,A) = r^* \cup sr^* \quad \text{and} \quad \mathcal{L}_\mathcal{T}(C,B) = ss^*.$$

Then, $\mathcal{L}_\mathcal{T}(C)$ is represented by the tree in Figure 3.10. ♢

As proved in [Pen15] (Theorem 5.21), $\mathcal{L}_\mathcal{T}(C,A)$ is a regular language for every concept name A. In fact, Pensel actually computes an rLTA[3] representing $\mathcal{L}_\mathcal{T}(C)$, that is, an LTA accepting exactly the infinite tree corresponding to the tuple of languages $\mathcal{L}_\mathcal{T}(C)$.

Theorem 4.6 ([Pen15]). *Given an $\mathcal{FL}_0$ concept description C and an $\mathcal{FL}_0$ TBox $\mathcal{T}$, an rLTA representing $\mathcal{L}_\mathcal{T}(C)$ can be constructed in time exponential in the size of C and $\mathcal{T}$. Furthermore, this rLTA provides a DFA of the same size for the language $\mathcal{L}_\mathcal{T}(C,A)$ for every $A \in \mathsf{N_C}$.*

Strictly speaking, the label set employed in [Pen15] is $2^{\mathsf{N_C}}$ for $\mathsf{N_C} = \{A_1,\dots,A_k\}$ rather than $\{0,1\}^k$, but it should be clear that, by fixing a linear order $A_1 < A_2 < \ldots < A_k$ on $\mathsf{N_C}$, these two representations can be translated into each other.

Using the same approach, given two concept descriptions C, D, the pair of tuples $(\mathcal{L}_\mathcal{T}(C), \mathcal{L}_\mathcal{T}(D))$ can be represented as an infinite $\mathsf{N_R}$-tree $t_{(C,D)}$ with label set $\{0,1\}^k \times \{0,1\}^k$. We can then use tree automata to check whether (the pair of tuples that corresponds to) such a tree satisfies the language inclusions $\mathcal{L}_\mathcal{T}(D,A) \subseteq \mathcal{L}_\mathcal{T}(C,A)$ for every $A \in \mathsf{N_C}$. In particular, it is easy to see that the LTA $\mathcal{A} = (\mathsf{N_R}, \{q\}, \{0,1\}^{2k}, \Delta, \{q\})$, where

$$\Delta = \{(q, \ell, q, \dots, q) \mid \ell = (x_1, \dots, x_k, x_{k+1}, \dots, x_{2k}) \text{ with } x_i \geq x_{k+i}\}$$

accepts exactly the pairs of tuples of languages with the aforementioned property. By taking the intersection of $\mathcal{A}$ with the (r)LTA representing $t_{(C,D)}$ and checking emptiness of the product automaton we can decide whether $C \sqsubseteq_\mathcal{T} D$ or not.

4.2 Unification

In order to define unification in $\mathcal{FL}_0$ with respect to an $\mathcal{FL}_0$ TBox, we first need to introduce the notions of *concept patterns* and *substitutions*. We consider an additional set $\mathsf{N_X}$, disjoint from $\mathsf{N_C}$ and $\mathsf{N_R}$, whose elements we call *concept variables*. Intuitively, $\mathsf{N_X}$ contains the concept names that have possibly been given another name or been specified in more detail in another concept description describing the same notion. From a syntactic point of view, concept variables are treated like concept names. More formally, the set $\mathcal{P}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R}, \mathsf{N_X})$ of *$\mathcal{FL}_0$ concept patterns over the concept names $\mathsf{N_C}$, the role names $\mathsf{N_R}$, and the concept variables $\mathsf{N_X}$* is the set $\mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C} \cup \mathsf{N_X}, \mathsf{N_R})$, i.e., a concept pattern is a concept description defined over the set of concept names $\mathsf{N_C} \cup \mathsf{N_X}$ (instead of simply $\mathsf{N_C}$) and the set of role names $\mathsf{N_R}$.

Informally, a unification problem in $\mathcal{FL}_0$ asks, given $\mathcal{FL}_0$ concept patterns C and D, whether the variables occurring in C and D can be replaced by concept descriptions such that the resulting expressions are equivalent w.r.t. a given $\mathcal{FL}_0$ TBox. The meaning of "replacing" in the previous sentence is formalized using the notion of a substitution. An *$\mathcal{FL}_0$ substitution σ from $\mathsf{N_X}$ to $\mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R})$* is a mapping assigning $\mathcal{FL}_0$ concept descriptions $\sigma(X) \in \mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R})$ to

[3] Even though the automaton is simply refered to as an LTA, a closer look reveals that, by construction, it is an rLTA.

variables $X \in \mathsf{N_X}$. The application of such a substitution σ to concept patterns is inductively defined as follows:

$$\sigma(\top) := \top, \quad \sigma(A) := A \text{ for all } A \in \mathsf{N_C},$$
$$\sigma(C \sqcap D) := \sigma(C) \sqcap \sigma(D), \quad \sigma(\forall r.C) := \forall r.\sigma(C).$$

We denote the set of all $\mathcal{FL}_0$ substitutions as $Sub(\mathsf{N_X}, \mathsf{N_C}, \mathsf{N_R})$, or simply *Sub* if there is no fear of confusion.

Definition 4.7 (Unification). *Let C, D be $\mathcal{FL}_0$ concept patterns, and $\mathcal{T}$ an $\mathcal{FL}_0$ TBox. The substitution σ is a* unifier *of C, D w.r.t. $\mathcal{T}$ if $\sigma(C) \equiv_{\mathcal{T}} \sigma(D)$. If C, D have a unifier, then we call them* unifiable *w.r.t. $\mathcal{T}$. The $\mathcal{FL}_0$* unification problem *asks whether two given $\mathcal{FL}_0$ concept patterns are unifiable w.r.t. a given TBox or not.* ♢

Given a unification problem $C \equiv^?_{\mathcal{T}} D$, only the σ-images of variables occurring in C or D are relevant. For this reason, we will assume in the following that $\mathsf{N_X}$ consists of exactly these variables.[4] Furthermore, let $\mathsf{N_C}$ and $\mathsf{N_R}$ consist of the concept and role names occurring in C and D. As proved in [BN01], it suffices to consider substitutions in $Sub(\mathsf{N_X}, \mathsf{N_C}, \mathsf{N_R})$, since if there exists a unifier, there exists one that only uses the signature occurring in the patterns.

Decidability of the $\mathcal{FL}_0$ unification problem with respect to an $\mathcal{FL}_0$ TBox is a long-standing open problem. The original paper on unification in $\mathcal{FL}_0$, [BN01], only deals with the special case of the empty TBox. Even for the dual logic $\mathcal{EL}$, where there have been considerably more results on unification, decidability in the presence of general TBoxes remains open.

In Chapter 8 we will try to tackle this problem, but we will only be able to integrate TBoxes for the case of *matching*. Matching is the special case of unification where one of the patterns to be unified contains no variables, i.e., it is a concept description, and is hence *invariant* under substitutions.

Example 4.8. *Consider the $\mathcal{FL}_0$ concept patterns $C = A \sqcap \forall r.(X \sqcap \forall s.B)$ and $D = A \sqcap \forall r.A \sqcap \forall r.Y$ (and an empty TBox). Intuitively, for σ to be a unifier the substitution for X has to make up for the value restriction $\forall r.A$ occurring in D, while the substitution for Y should ensure that the restriction $\forall r.\forall s.B$ occurs in $\sigma(D)$. It is easy to verify that the substitution $\sigma = \{X \mapsto A, Y \mapsto \forall s.B\}$ is a unifier, since*

$$\sigma(C) = A \sqcap \forall r.(A \sqcap \forall s.B) \equiv A \sqcap \forall r.A \sqcap \forall r.\forall s.B = \sigma(D).$$ ♢

Size Before we talk about complexity of unification problems, we should also define the size of the involved notions. In particular, the size of a concept pattern is defined in the same way as the size of a concept description by considering concept variables as concept names, i.e., by setting $|X| = 1$ for every $X \in \mathsf{N_X}$.[5] The size of a unification problem $C =^? D$ is defined to be the sum of the sizes of the concept patterns C and D, plus the size of the given TBox $\mathcal{T}$.[6]

[4] Also recall our discussion on $\mathsf{N_C}$ and $\mathsf{N_R}$.

[5] Note that, in particular, if a concept description is viewed as a pattern, its size remains the same.

[6] Since we can restrict our focus to the concept and role names occurring in the problem, we do not have to worry about the size of $\mathsf{N_C}$ and $\mathsf{N_R}$.

Furthermore, the size of a substitution is defined to be the sum of the sizes of the concept descriptions that substitute the concept variables, in symbols $|\sigma| = \sum_{X \in \mathsf{N_X}} |\sigma(X)|$.

We will next introduce the approach for $\mathcal{FL}_0$ unification with respect to the empty TBox. For the sake of readability, we will simply refer to this as the unification problem for $\mathcal{FL}_0$, not mentioning that the TBox is empty.

4.3 Reducing unfication to language equations

In this section we will recall the approach of Baader and Narendran, who showed in [BN01] that the $\mathcal{FL}_0$ unification problem (w.r.t. the empty TBox) is ExpTime-complete. The ExpTime upper bound is proved by a reduction to language equations, which in turn are solved using tree automata. Conversely, a reduction from a tree automata problem is used to prove the lower bound.[7] Here we sketch the reduction to language equations. In the next section we will discuss how solvability of these language equations can be checked.

Let C, D be $\mathcal{FL}_0$ concept patterns, and let $\mathsf{N_C} = \{A_1, \ldots, A_k\}$, $\mathsf{N_R}$, $\mathsf{N_X} = \{X_1, \ldots, X_m\}$ respectively be the sets of concept names, role names, and concept variables occurring in C or D. If we treat concept variables like concept names we can apply the normal form (4.2) to patterns as well. Hence, we can assume without loss of generality that the input patterns are in normal form, i.e.,

$$\begin{aligned} C &= \forall K_{0,1}.A_1 \sqcap \ldots \sqcap \forall K_{0,k}.A_k \sqcap \forall K_1.X_1 \sqcap \ldots \sqcap \forall K_m.X_m, \\ D &= \forall L_{0,1}.A_1 \sqcap \ldots \sqcap \forall L_{0,k}.A_k \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_m.X_m, \end{aligned} \tag{4.3}$$

where $K_{0,i}, L_{0,i}, K_j, L_j$ are finite languages over $\mathsf{N_R}$. The unification problem for C, D can be reduced to (independently) solving the language equations

$$K_{0,i} \cup K_1 \cdot X_{1,i} \cup \ldots \cup K_m \cdot X_{m,i} \stackrel{?}{=} L_{0,i} \cup L_1 \cdot X_{1,i} \cup \ldots \cup L_m \cdot X_{m,i} \tag{4.4}$$

for $i = 1, \ldots, k$, where "$\cdot$" stands for concatenation of languages. A *solution* σ_i of such an equation is an *assignment* of languages (over $\mathsf{N_R}$) to the variables $X_{j,i}$ such that

$$K_{0,i} \cup K_1 \cdot \sigma_i(X_{1,i}) \cup \ldots \cup K_m \cdot \sigma_i(X_{m,i}) = L_{0,i} \cup L_1 \cdot \sigma_i(X_{1,i}) \cup \ldots \cup L_m \cdot \sigma_i(X_{m,i}).$$

This assignment is called *finite* if all the languages $\sigma_i(X_{j,i})$ are finite. We denote the set of all assignments as *Ass* and the set of all finite assignments as *finAss*.

As shown in [BN01], C, D are unifiable iff the language equations of the form (4.4) have finite solutions for all $i = 1, \ldots, k$. In fact, given finite solutions $\sigma_1, \ldots, \sigma_k$ of these equations, a unifier of C, D can be obtained by setting

$$\sigma(X_i) := \forall \sigma_i(X_{i,1}).A_1 \sqcap \ldots \sqcap \forall \sigma_i(X_{i,k}).A_k, \tag{4.5}$$

and every unifier of C, D can be obtained in this way. Of course, this construction of a substitution from a k-tuple of finite assignments can be applied to arbitrary finite assignments

[7] This reduction requires concept patterns containing arbitrarily many role names and concept variables, but only a single concept name. Whether the problem is ExpTime-hard for a fixed set of role names is, to the best of our knowledge, not known.

(and not just to finite solutions of the equations (4.4)), and it yields a bijection ρ between k-tuples of finite assignments and substitutions.

Coming back to Example 4.8, the given concept patterns have normal forms

$$C = \forall\{\varepsilon\}.A \sqcap \forall\{rs\}.B \sqcap \forall\{r\}.X \sqcap \forall\emptyset.Y$$
$$D = \forall\{\varepsilon, r\}.A \sqcap \forall\emptyset.B \sqcap \forall\emptyset.X \sqcap \forall\{r\}.Y.$$

Hence, the language equations for the concept names A and B are

$$\{\varepsilon\} \cup \{r\} \cdot X_A \cup \emptyset \cdot Y_A =^? \{\varepsilon, r\} \cup \emptyset \cdot X_A \cup \{r\} \cdot Y_A,$$
$$\{rs\} \cup \{r\} \cdot X_B \cup \emptyset \cdot Y_B =^? \emptyset \cup \emptyset \cdot X_B \cup \{r\} \cdot Y_B.$$

Among others, the first equation has $X_A = \{\varepsilon\}, Y_A = \emptyset$ as a solution, and the second $X_B = \emptyset$ and $Y_B = \{s\}$. Using (4.5), but leaving out the value restrictions for $\emptyset$ (that would denote $\top$ anyway), these solutions yield the unifier σ that we came up with in Example 4.8.

Another solution to the above language equations is $X_A = Y_A = \{\varepsilon\}$ and $X_B = Y_B = \{\varepsilon, s\}$, which yields the unifier σ' with $\sigma'(X) = \sigma'(Y) = A \sqcap B \sqcap \forall s.B$.

4.4 Solving language equations

In order to check solvability of language equations of the above form, Baader and Narendran [BN01] utilize automata on finite trees. As they pointed out, however, it is not clear how tree automata can deal directly with language equations of the form (4.4). In particular, the variables X_i should appear *in front* of the coefficients K_i, L_i if such automata are to be used. Yet, such an equation can easily be obtained from 4.4 by considering the mirror images of the involved languages. Recall that, for a word $w = \sigma_1 \dots \sigma_\ell$, its mirror image is defined as $w^{mi} = \sigma_\ell \dots \sigma_1$, and for a language L its mirror image is $L^{mi} = \{w^{mi} \mid w \in L\}$. Obviously, σ is a solution of (4.4) iff $\sigma' = \{Y_i \mapsto \sigma(X_i)^{mi} \mid i = 1, \dots, m\}$ is a solution of the corresponding mirrored equation:

$$K_0^{mi} \cup Y_1 K_1^{mi} \cup \dots \cup Y_m K_m^{mi} =^? L_0^{mi} \cup Y_1 L_1^{mi} \cup \dots \cup Y_m L_m^{mi}. \tag{4.6}$$

If σ' is a finite assignment given by enumeration of the words occuring in the finite languages, σ is immediately derivable and of the same size. By Lemma 3.5, if it is given as a tuple of DFAs of size n, DFAs of size 2^n are required for the mirror languages, while the number drops to $n+1$ when we consider NFAs.

In this section, following [BO13], we consider more general forms of language equations than the ones given in (4.4) and (4.6). Here, all Boolean operators (and not just union) are available. *Language expressions* are built recursively over a finite alphabet Σ using union, intersection, complement, and one-sided concatenation of regular languages. Language expressions with concatenation of regular languages from the left are formalized by the syntax rules:

$$\phi ::= L \mid X \mid \phi \cup \phi \mid \phi \cap \phi \mid \sim\phi \mid L \cdot \phi, \tag{4.7}$$

while if concatenation is applied from the right we have:

$$\phi ::= L \mid X \mid \phi \cup \phi \mid \phi \cap \phi \mid \sim\phi \mid \phi \cdot L. \tag{4.8}$$

In both cases, L can be instantiated with any finite language over Σ and X with any variable. Obviously, the left- and the right-hand sides of (4.4) are language expressions of the form (4.7), while for the mirrored equation (4.6) of the form (4.8). As before, an *assignment* $\sigma \in Ass$ maps variables to languages over Σ. It is extended to expressions in the expected way, that is $\sigma(L) := L$, $\sigma(\phi \cup \psi) := \sigma(\phi) \cup \sigma(\psi)$, $\sigma(\phi \cap \psi) := \sigma(\phi) \cap \sigma(\psi)$, $\sigma(\sim\phi) := \sigma(\phi)^c$, $\sigma(L \cdot \phi) := L\sigma(\phi)$, $\sigma(\phi \cdot L) := \sigma(\phi)L$.

Definition 4.9 (Solvability of language equations). *Let ϕ, ψ be language expressions that are both of the form* (4.7) *or both of the form* (4.8)*. The assignment σ* solves *the* language equation $\phi =^? \psi$ *if $\sigma(\phi) = \sigma(\psi)$. For* finite solvability *we require the languages $\sigma(X)$ to be finite, i.e., σ should be an element of finAss.* ♢

The argument from the previous section about mirrored equations applies here as well. More formally, we have the following.

Lemma 4.10. *For every language equation between language expressions of the form* (4.7) *we can construct in linear time a language equation between language expressions of the form* (4.8)*, such that there is a one-to-one correspondence between the solutions of the one and the solutions of the other. In particular, this implies that solvability of the one implies solvability of the other. The converse is also true.*

To see why this holds, it suffices to extend mirroring to expressions and assignments. For expressions this can be done inductively, and it is straightforward; in particular, we have $(L \cdot \phi)^{mi} = \phi^{mi} \cdot L^{mi}$ and vice versa, while we assume that $X^{mi} = X$. Given an assignment $\sigma = \{X_i \mapsto \sigma(X_i) \mid i = 1, \ldots, m\}$, define $\sigma^{mi} = \{X_i \mapsto \sigma(X_i)^{mi} \mid i = 1, \ldots, m\}$. These definitions suffice to verify that, if $\phi =^? \psi$ is a language equation between expressions that are both of one of the two forms, then $\phi^{mi} =^? \psi^{mi}$ contains expressions of the other form, and σ is a solution of the first iff σ^{mi} is a solution of the second.

Hence, it does not matter if we are exploring solvability of language equations between expressions with concatenation from the left or from the right. Following [BO13], we deal with the latter.

The first step to check solvability is to transform the given language equation $\phi_1 =^? \phi_2$ into one of the form $\phi =^? \emptyset$ such that the language expression ϕ is *normalized* in the sense that all constant languages L occurring in ϕ are singleton languages $\{a\}$ for $a \in \Sigma \cup \{\varepsilon\}$. Basically, this approach introduces new variables and equations that express the constant languages occurring in ϕ_1 and ϕ_2, using the fact that these languages can be expressed as unique solutions of language equations in solved form (see [BO13] for details). The resulting system of equations can then be expressed by a single equation $\phi =^? \emptyset$ using the facts that

- $M = N$ iff $M \triangle N = \emptyset$, where $M \triangle N$ is an abbreviation for $(M \cap \sim N) \cup (N \cap \sim M)$;
- $M = \emptyset$ and $N = \emptyset$ iff $M \cup N = \emptyset$.

Lemma 4.11 ([BO13], Lemma 1). *Let $\phi_1 =^? \phi_2$ be a language equation between expressions both of the form* (4.7) *or* (4.8)*. Then we can compute in polynomial time a normalized language*

expression ϕ_0 (over an extended set of variables) such that there is a one-to-one correspondence between the solutions of $\phi_1 =^? \phi_2$ and the solutions of $\phi =^? \emptyset$.

In a second step, [BO13] shows how a normalized language equation can be translated into an LTA working on the infinite, unlabeled n-ary tree (where $n = |\Sigma|$). Recall that the nodes of this tree can be identified with Σ^*.

Since runs of an LTA are themselves trees with the set of states as the label set, using the procedure we introduced in Definition 3.22, i.e., by designating a set of states F, a run r of the LTA $\mathcal{A}$ defines the language

$$L_r(\mathcal{A}, F) := \{w \in \Sigma^* \mid r(w) \in F\}.$$

Given a normalized language equation $\phi =^? \emptyset$ with variables $\{X_1, \ldots, X_m\}$, it is shown in [BO13] how to construct an LTA $\mathcal{A}^\phi = (\Sigma, Q^\phi, \{\ell\}, \Delta^\phi, I^\phi)$ and subsets $F_1, \ldots, F_m \subseteq Q^\phi$ such that the following holds:

Proposition 4.12. *Given a run r of $\mathcal{A}^\phi$, define the* induced assignment *σ_r by setting $\sigma_r(X_i) := L_r(\mathcal{A}^\phi, F_i)$, for $i = 1, \ldots, m$. Then, for every subexpression ψ of ϕ there exists a set $F_\psi \subseteq Q$ such that $\sigma_r(\psi) = L_r(\mathcal{A}^\phi, F_\psi)$. In addition, every assignment is induced by some run of $\mathcal{A}_\phi$.* ♢

The size of this LTA is exponential in the size of ϕ. In order to decide whether the language equation $\phi =^? \emptyset$ has a solution, one thus needs to decide whether $\mathcal{A}^\phi$ has a run in which no state of F_ϕ occurs. This can easily be done by removing all states of F_ϕ from $\mathcal{A}^\phi$, and then checking the resulting automaton $\mathcal{A}^{-\phi}$ for emptiness. In fact, as an easy consequence of the above proposition we obtain that there is a one-to-one correspondence between the runs of $\mathcal{A}^{-\phi}$ and the solutions of $\phi =^? \emptyset$ (Proposition 2 in [BO13]).

The above approach, however, as described until now can only check (general) solvability of language equations, and not finite solvability, which is required for the original problem of unification in $\mathcal{FL}_0$.

Let us call the states in $\bigcup_{i=1}^m F_i$ *variable states*. If a run of $\mathcal{A}^{-\phi}$ assigns such a state q to a word $w \in \Sigma^*$, this means that the induced assignment for the variable(s) corresponding to q contains w.

Hence, every run of $\mathcal{A}^{-\phi}$ that uses variable states only finitely often induces a finite solution of the original language equation, and conversely, every finite solution of the language equation is induced by such a run. Therefore, in order to check whether there are finite solutions it suffices to check whether $\mathcal{A}^{-\phi}$ has a run that uses variable states only finitely often.

For this purpose we can implement Lemma 3.43 by setting $B = \bigcup_{i=1}^m F_i$. Since $\mathcal{A}^{-\phi}$ is exponential in the size of the language equation, we obtain an ExpTime procedure also for checking finite solvability.

Chapter 5

Concept Distances

The goal of approximate unification is to increase the recall of classical unification by no longer requiring two concepts to become equivalent, but rather *"almost"* equivalent. In this chapter we formalize the notion of *almost* by introducing functions that measure the distance between $\mathcal{FL}_0$ concept descriptions in the presence of background knowledge, namely a general TBox.

Concept distance measures (CDMs) are, in a sense, the generalization of metrics in the setting of knowledge representation. Fundamentaly, such a measure quantifies how close two objects are from a conceptual point of view. A concept distance measure takes as input two concept descriptions and outputs, in the most general setting, a non-negative real number, with 0 standing for "no distance", that is, complete similarity. *Normalized distance measures* restrict the output to a value between 0 and 1, with 1 standing for the maximum distance possible, that is, total dissimilarity. More generally, the smaller the outputed number is, the more "close together" the input concept descriptions are. However, it is *concept similarity measures (CSMs)*, the dual notion of CDMs, that have received considerably more interest in the literature. A concept similarity measure usually outputs a number between 0 and 1, but here the semantics is reversed: it is 1 standing for complete similarity between the concepts and 0 for total dissimilarity. More generally, a larger number indicates a greater degree of similarity. Note that we can define a (normalized) concept distance measure d_s from a similarity measure s by setting $d(C,D) = 1 - s(C,D)$, and vice versa.[1] We will refer to both kinds of measures indistinguishably as *concept comparison measures (CCMs)*. CCMs are an integral part of approximate reasoning services, as we discussed shortly in the Introduction of the thesis. The need for such measures is highlighted and addressed in the PhD thesis of Ecke [Eck17], where furthermore concrete examples are provided on how to integrate them in the formalism of Description Logics.

As we mentioned in the introduction of the thesis, research on concept comparison measures in DLs started with [BWH05] and the very inexpressive logic $\mathcal{A}$, and until recently, it was mostly concerned with $\mathcal{EL}$ [LT12; EPT15; Sun13] and more expressive DLs [dFE05]. Even though the exact definition might vary slightly from one paper to another, all approaches agree on the fact that the measure should respect the semantics of the formalism. In particular, equivalence of concepts should be reflected by the measure, that is the distance between equivalent concepts should be minimal and equivalent concepts should be equidistant from other concepts. To ensure this, concept descriptions are usually first translated into an appropriate normal form, and then the structure of the normalized descriptions is compared.

[1] Further ways of transiting from one setting to the other have been considered in the literature. In particular, another way to obtain a similarity measure s from a (not necessarily normalized) distance measure d is to set $s(C,D) = e^{-d(C,D)}$ [She87].

For instance, measures that satisfy this condition for the empty TBox or for acyclic TBoxes in $\mathcal{EL}$ [LT12; Sun13] make use of the reduced form of $\mathcal{EL}$ concept descriptions introduced in [Küs01]. Extensions to general TBoxes [EPT15] use the so-called canonical model, which is generated by the polynomial-time subsumption algorithm for $\mathcal{EL}$ [BBL05].

A recent approach for defining concept comparison measures for $\mathcal{FL}_0$ [RS15] was restricted to the case of the empty TBox. Unsurprisingly, it employs the formal language-based characterization of equivalence between $\mathcal{FL}_0$ concept descriptions (with respect to the empty TBox) that we described in the previous chapter.

In the remainder of this chapter, we will develop a general approach for defining concept distance measures for $\mathcal{FL}_0$ concept descriptions that respect the semantics also w.r.t. general TBoxes. This is achieved by using the characterization of equivalence in $\mathcal{FL}_0$ w.r.t. general TBoxes (Proposition 4.4). Since equivalent concept descriptions correspond to the same tuple of languages, it is sufficient to define measures that compare such tuples.

After we describe this framework and give concrete examples, we will introduce properties for CDMs, originating in [LT12], and investigate which of them our measures satisfy.

5.1 Formal definitions

We now formally define CDMs and what it means to respect the semantics of DLs.

Definition 5.1. *A* concept distance measure *for $\mathcal{FL}_0$ is a family of functions m that contains for every general $\mathcal{FL}_0$ TBox $\mathcal{T}$ a function $m_\mathcal{T} : \mathcal{C}_{\mathcal{FL}_0} \times \mathcal{C}_{\mathcal{FL}_0} \to [0, +\infty)$ that for every $C, D, E \in \mathcal{C}_{\mathcal{FL}_0}$ satisfies the following properties*

- *equivalence closedness:* $m_\mathcal{T}(C, D) = 0 \iff C \equiv_\mathcal{T} D$,
- *symmetry:* $m_\mathcal{T}(C, D) = m_\mathcal{T}(D, C)$,
- *equivalence invariance:* $C \equiv_\mathcal{T} D \implies m_\mathcal{T}(C, E) = m_\mathcal{T}(D, E)$.

If the requirement for finiteness is dropped, that is if we allow the distance between concepts to become infinite, we have a generalized CDM*, while if we restrict the value to the interval $[0, 1]$, we obtain a* normalized CDM. ♢

Note that equivalence closedness corresponds to $(M1)$ and symmetry to $(M2)$ in the definition of a metric. Equivalence invariance is a condition *sine qua non*; it is the essence of *respecting the semantics* and it is ubiquitous, for it appears in (almost) every paper that treats CCMs.

The notion of CDM generalizes equivalence. In fact, equivalence can be seen as the very simple CDM m that is defined as $m_\mathcal{T}(C, D) = 0$ if $C \equiv_\mathcal{T} D$ and 1 otherwise. It is immediately verifiable that this is in fact a normalized CDM.

In order to define more meaningful CDMs, we will employ the value restriction sets introduced in Section 4.1. Lemma 4.4 shows that, in $\mathcal{FL}_0$, these formal languages can be used to represent the semantic content of concept descriptions: up to equivalence, every $\mathcal{FL}_0$ concept description $C \in \mathcal{C}_{\mathcal{FL}_0}(\mathsf{N_C}, \mathsf{N_R})$ is uniquely represented by the tuple of languages

$$\mathcal{L}_\mathcal{T}(C) := (\mathcal{L}_\mathcal{T}(C, A_1), \ldots, \mathcal{L}_\mathcal{T}(C, A_k)).$$

We will use this fact to reduce the definition of concept comparison measures between $\mathcal{FL}_0$ concept descriptions w.r.t. a TBox to the definition of measures comparing tuples of languages: given two $\mathcal{FL}_0$ concept descriptions C, D, we define $\mathfrak{c}_{\mathcal{T}}(C, D)$ by comparing the tuples $\mathcal{L}_{\mathcal{T}}(C)$ and $\mathcal{L}_{\mathcal{T}}(D)$. One advantage of this approach is that equivalence invariance comes "for free" since equivalent concept descriptions are indistinguishable from the language point of view.

5.2 Using tuples of languages to define CDMs

The idea of using tuples of languages to compare $\mathcal{FL}_0$ concept descriptions has already appeared in [RS15], but restricted to the empty TBox. Generally speaking, the approach used consists of the following three steps:

1. Translate the $\mathcal{FL}_0$ concept descriptions C and D into their corresponding tuples of languages $\mathcal{L}_\emptyset(C) = (K_1, \ldots, K_\ell)$ and $\mathcal{L}_\emptyset(D) = (L_1, \ldots, L_\ell)$. For the sake of readability, we will denote these tuples as $\mathbf{K}$ and $\mathbf{L}$, respectively.
2. To compare the tuples $\mathbf{K}$ and $\mathbf{L}$, their components K_i and L_i are compared pairwise, and the values obtained this way are then appropriately combined into a value $s(\mathbf{K}, \mathbf{L})$.
3. Finally, the value $s(\mathbf{K}, \mathbf{L})$ is used to define $\mathfrak{c}_\emptyset(C, D)$.

Essentially, the definition of the measure is in the end reduced to define a function that compares two languages. We now apply a similar technique in order to define CDMs in the presence of general TBoxes. In particular, given a TBox $\mathcal{T}$ and concept descriptions C, D, we first use a language distance d to compute for every concept name $A_i \in \mathsf{N_C}$ the value $e_i = d(\mathcal{L}_{\mathcal{T}}(C, A_i), \mathcal{L}_{\mathcal{T}}(D, A_i))$. We then combine these k values to define a distance between C and D. For this, we need an appropriate function. We say that the function $f : [0, \infty)^k \to [0, \infty)$ is a *combining function* if it is

- monotone: $a_1 \leqslant b_1, \ldots, a_k \leqslant b_k \implies f(a_1, \ldots, a_k) \leqslant f(b_1, \ldots, b_k)$,
- zero-sum free: $f(a_1, \ldots, a_k) = 0 \iff a_1 = \cdots = a_k = 0$,
- and continuous.

The following are simple examples of combining functions:

- $\max(a_1, \ldots, a_k)$,
- $\mathrm{sum}(a_1, \ldots, a_k) = a_1 + \cdots + a_k$,
- $\mathrm{avg}(a_1, \ldots, a_k) = \sum_{i=1}^{k} a_i / k$.

As we did for CDMs and language distances, we also introduce the notion of *generalized combining functions* by considering functions capable of handling infinity, i.e., functions $f : [0, \infty]^k \to [0, \infty]$ satisfying the above conditions, where by $[0, \infty]$ we denote the set $[0, +\infty) \cup \{+\infty\}$. Note that all of the functions we provided as examples can also operate as generalized combining functions, if we set $f(a_1, \ldots, a_k) = +\infty$ in case (at least) one of the arguments is $+\infty$.

We are now ready to formally define how to obtain a CDM starting from a language distance.

Definition 5.2. *Given a language distance d and a combining function f, we define the* measure $m^{d,f}$ induced by d and f *by setting for every $\mathcal{FL}_0$ TBox $\mathcal{T}$ and concept descriptions C, D:*

$$m^{d,f}_{\mathcal{T}}(C,D) = f\big(d(\mathcal{L}_{\mathcal{T}}(C,A_1),\mathcal{L}_{\mathcal{T}}(D,A_1)),\ldots,d(\mathcal{L}_{\mathcal{T}}(C,A_k),\mathcal{L}_{\mathcal{T}}(D,A_k))\big).$$

♢

It is easy to verify that this construction fulfills its purpose, as the following lemma demonstrates.

Lemma 5.3. *Let d be a language distance and f be a combining function. Then the measure $m^{d,f}$ induced by d, f is a CDM, i.e., it is equivalence closed, symmetric, and equivalence invariant.*

Proof. Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox, and C, D, E be $\mathcal{FL}_0$ concept descriptions. All properties can be proved with simple computations.
Equivalence closedness:

$$\begin{aligned} m^{d,f}_{\mathcal{T}}(C,D) = 0 &\iff f\big(d(\mathcal{L}_{\mathcal{T}}(C,A_1),\mathcal{L}_{\mathcal{T}}(D,A_1)),\ldots,d(\mathcal{L}_{\mathcal{T}}(C,A_k),\mathcal{L}_{\mathcal{T}}(D,A_k))\big) = 0 \\ &\overset{(1)}{\iff} d(\mathcal{L}_{\mathcal{T}}(C,A_1),\mathcal{L}_{\mathcal{T}}(D,A_1)) = \cdots = d(\mathcal{L}_{\mathcal{T}}(C,A_k),\mathcal{L}_{\mathcal{T}}(D,A_k)) = 0 \\ &\overset{(2)}{\iff} \mathcal{L}_{\mathcal{T}}(C,A_1) = \mathcal{L}_{\mathcal{T}}(D,A_1),\ldots,\mathcal{L}_{\mathcal{T}}(C,A_k) = \mathcal{L}_{\mathcal{T}}(D,A_k) \\ &\overset{(3)}{\iff} C \equiv_{\mathcal{T}} D, \end{aligned}$$

where (1) holds because f is zero-sum free, (2) by d being a metric, and hence satisfying property $(M1)$, and (3) by Lemma 4.4.
Symmetry:

$$\begin{aligned} m^{d,f}(C,D) &= f\big(d(\mathcal{L}_{\mathcal{T}}(C,A_1),\mathcal{L}_{\mathcal{T}}(D,A_1)),\ldots,d(\mathcal{L}_{\mathcal{T}}(C,A_k),\mathcal{L}_{\mathcal{T}}(D,A_k))\big) \\ &= f\big(d(\mathcal{L}_{\mathcal{T}}(D,A_1),\mathcal{L}_{\mathcal{T}}(C,A_1)),\ldots,d(\mathcal{L}_{\mathcal{T}}(D,A_k),\mathcal{L}_{\mathcal{T}}(C,A_k))\big) = m^{d,f}(D,C), \end{aligned}$$

since d is a metric, hence satisfying property $(M2)$.
Equivalence invariance:
$C \equiv_{\mathcal{T}} D \overset{\text{Lemma 4.4}}{\Longrightarrow} \mathcal{L}_{\mathcal{T}}(C,A_i) = \mathcal{L}_{\mathcal{T}}(D,A_i)$ for $i = 1,\ldots,k$.

$$\begin{aligned} m^{d,f}(C,E) &= f\big(d(\mathcal{L}_{\mathcal{T}}(C,A_1),\mathcal{L}_{\mathcal{T}}(E,A_1)),\ldots,d(\mathcal{L}_{\mathcal{T}}(C,A_k),\mathcal{L}_{\mathcal{T}}(E,A_k))\big) \\ &= f\big(d(\mathcal{L}_{\mathcal{T}}(D,A_1),\mathcal{L}_{\mathcal{T}}(E,A_1)),\ldots,d(\mathcal{L}_{\mathcal{T}}(D,A_k),\mathcal{L}_{\mathcal{T}}(E,A_k))\big) = m^{d,f}(D,E). \quad \square \end{aligned}$$

Note that if d is a generalized language distance and f a generalized combining function, the exact proof from above can be used to prove that $m^{d,f}$ is a generalized CDM.

Furthermore, the keen reader might have noticed that continuity of f has not been used yet. Indeed, this property is not needed to ensure that $m^{d,f}$ is a CDM. However, it is a very natural requirement for a function to be "well-behaved", and it will be essential when we reduce the problem of approximate unification to approximate solvability of language equations in Chapter 6.

5.3 Some instances of CDMs

Let us now instantiate the language distance d of the framework we introduced above with the distances d_0, d_1 and d_2 we investigated in Section 3.3. All three functions are based on the symmetric difference of the languages under consideration. Recall that, given formal languages M_1 and M_2 over the alphabet Σ their symmetric difference $M_1 \Delta M_2$ is defined to be

$$M_1 \Delta M_2 := (M_1 \setminus M_2) \cup (M_2 \setminus M_1).$$

The language distances d_0, d_1 and d_2 are defined as

$$\begin{aligned} d_0(M_1, M_2) &:= |M_1 \Delta M_2|, \\ d_1(M_1, M_2) &:= 2^{-n} && \text{where } n = \min\{|w| \mid w \in M_1 \Delta M_2\}, \\ d_2(M_1, M_2) &:= \mu(M_1 \Delta M_2) && \text{where } \mu(M) = \tfrac{1}{2} \sum_{w \in M} (2|\Sigma|)^{-|w|}. \end{aligned}$$

Intuitively, the symmetric difference captures all the discrepancies between two concept descriptions C and D with respect to a concept name A. More precisely, if for instance, $w \in \mathcal{L}_\mathcal{T}(C, A) \setminus \mathcal{L}_\mathcal{T}(D, A)$ for some $w \in \mathsf{N_R}^*$, then $C \sqsubseteq_\mathcal{T} \forall w.A$ and $D \not\sqsubseteq_\mathcal{T} \forall w.A$, which amounts to a semantically relevant difference between C and D. Based on this intuition, d_0 simply counts all such discrepancies. Note that, formally, d_0 is a generalized language distance, and its value may become infinite, since value restriction sets are usually infinite languages. It hence induces a generalized CDM that is very straightforward, but leaves a lot to be desired. Using one of the language distances d_1, d_2 in this setting means that differences between the concepts at larger role depth count less than differences at smaller role depth. From a semantics point of view this is a quite valid requirement, since differences in the properties of two given individuals and of their near successors should bear more importance when computing their difference than the properties of their more distant descendants. The distance d_1 looks for the shortest such discrepancy, while d_2 takes all differences into account, but differences for longer words count less than differences for shorter ones. Since these functions output a number between 0 and 1 as distance, using a combining function like max or avg would result in $m^{d,f}$ being a normalized CDM. In particular, this implies that, by taking $1 - m^{d,f}$, we obtain a CSM, as we described in the introduction of this chapter.

5.4 Further properties for CDMs

In the prominent work of Lehmann and Turhan [LT12], a few more properties on concept measures were investigated. Next, we have a look at some of these properties and investigate to what extent CDMs obtained from our framework satisfy them.

Definition 5.4. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox C, D, E be $\mathcal{FL}_0$ concept descriptions. We say that the CDM m is*

- *subsumption preserving if $C \sqsubseteq_\mathcal{T} D \sqsubseteq_\mathcal{T} E \implies m_\mathcal{T}(C, D) \leq m_\mathcal{T}(C, E)$.*
- *reverse subsumption preserving if $C \sqsubseteq_\mathcal{T} D \sqsubseteq_\mathcal{T} E \implies m_\mathcal{T}(D, E) \leq m_\mathcal{T}(C, E)$.*
- *fulfilling the* triangle inequality *if $m_\mathcal{T}(C, E) \leq m_\mathcal{T}(C, D) + m_\mathcal{T}(D, E)$.* ♢

Note that in [LT12], subsumption was considered with respect to the empty or an unfoldable TBox, and the properties were considered for similarity measures. Extending to the case of general TBoxes is straightforward, and the translation of the properties in the setting of distance measures is easy to obtain given the discussion in the beginning of this chapter. For an in-depth analysis of these properties and a nice review of the literature, the interested reader should look into [LT12; Leh12].

We will now prove that for language distances satisfying property (3.1) introduced in Section 3.3, that is if $K \Delta L \subseteq M \Delta N \implies d(K,L) \leq d(M,N)$, the CDM $m^{d,f}$ is subsumption preserving and reverse subsumption preserving.

More precisely, we have the following.

$$
\begin{aligned}
C \sqsubseteq_{\mathcal{T}} D \sqsubseteq_{\mathcal{T}} E \quad & \overset{(1)}{\implies} \mathcal{L}_{\mathcal{T}}(C,A_i) \supseteq \mathcal{L}_{\mathcal{T}}(D,A_i) \supseteq \mathcal{L}_{\mathcal{T}}(E,A_i) \text{ for every } i = 1,\dots,k \\
& \overset{(2)}{\implies} \mathcal{L}_{\mathcal{T}}(C,A_i) \Delta \mathcal{L}_{\mathcal{T}}(D,A_i) \subseteq \mathcal{L}_{\mathcal{T}}(C,A_i) \Delta \mathcal{L}_{\mathcal{T}}(E,A_i) \text{ and} \\
& \quad \mathcal{L}_{\mathcal{T}}(D,A_i) \Delta \mathcal{L}_{\mathcal{T}}(E,A_i) \subseteq \mathcal{L}_{\mathcal{T}}(C,A_i) \Delta \mathcal{L}_{\mathcal{T}}(E,A_i) \text{ for every } i = 1,\dots,k \\
& \overset{(3)}{\implies} d(\mathcal{L}_{\mathcal{T}}(C,A_i), \mathcal{L}_{\mathcal{T}}(D,A_i)) \leq d(\mathcal{L}_{\mathcal{T}}(C,A_i), \mathcal{L}_{\mathcal{T}}(E,A_i)) \text{ and} \\
& \quad d(\mathcal{L}_{\mathcal{T}}(D,A_i), \mathcal{L}_{\mathcal{T}}(E,A_i)) \leq d(\mathcal{L}_{\mathcal{T}}(C,A_i), \mathcal{L}_{\mathcal{T}}(E,A_i)) \text{ for every } i = 1,\dots,k \\
& \overset{(4)}{\implies} m^{d,f}_{\mathcal{T}}(C,D) \leq m^{d,f}_{\mathcal{T}}(C,E) \text{ and } m^{d,f}_{\mathcal{T}}(D,E) \leq m^{d,f}_{\mathcal{T}}(C,E),
\end{aligned}
$$

where (1) holds by Lemma 4.4, (2) is an easy set theoretic excercise, (3) holds because we assumed that d satisfies property (3.1), and (4) holds since f is monotone.

For the triangle inequality to hold, we need to impose a constraint on the combining function, namely restrict to *subadditive* functions. We say that a function $f : A \to \mathbb{R}$, with $A \subseteq \mathbb{R}^k$, is subadditive if for every $a, b \in A$ it holds that $f(a+b) \leq f(a) + f(b)$. Note that all three examples we provided as combining functions are actually subadditive. Furthermore, any concave function is also subadditive [Rud87].

We will now show that if f is a subadditive combining function, $m^{d,f}$ fulfills the triangle inequality for any language distance d. Since d is in particular a metric, we know that for every $\mathcal{FL}_0$ concept descriptions C, D, E and concept name A_i it holds that

$$
d(\mathcal{L}_{\mathcal{T}}(C,A_i), \mathcal{L}_{\mathcal{T}}(E,A_i)) \leq d(\mathcal{L}_{\mathcal{T}}(C,A_i), \mathcal{L}_{\mathcal{T}}(D,A_i)) + d(\mathcal{L}_{\mathcal{T}}(D,A_i), \mathcal{L}_{\mathcal{T}}(E,A_i)). \tag{5.1}
$$

For the sake of readability, for every $i = 1,\dots,k$ we set $c_i = d(\mathcal{L}_{\mathcal{T}}(C,A_i), \mathcal{L}_{\mathcal{T}}(E,A_i))$, $d_i = d(\mathcal{L}_{\mathcal{T}}(C,A_i), \mathcal{L}_{\mathcal{T}}(D,A_i))$, $e_i = d(\mathcal{L}_{\mathcal{T}}(D,A_i), \mathcal{L}_{\mathcal{T}}(E,A_i))$. Hence, (5.1) can be written as $c_i \leq d_i + e_i$. By monotonicity of f we have

$$
f(c_1,\dots,c_k) \leq f(d_1 + e_1,\dots,d_k + e_k).
$$

Finally, since f is subadditive, we have

$$
f(d_1 + e_1,\dots,d_k + e_k) \leq f(d_1,\dots,d_k) + f(e_1,\dots,e_k),
$$

which is exactly $m^{d,f}_{\mathcal{T}}(C,E) \leq m^{d,f}_{\mathcal{T}}(C,D) + m^{d,f}_{\mathcal{T}}(D,E)$.

A further property that is discussed in [LT12] is *structural dependency*, that intuitively requires that the more common attributes the two concepts share, the closer together they

should be. Formally, in our setting this is translated as follows: a CDM m is structurally dependent if for every TBox $\mathcal{T}$ and for all sequences $(C_n)_n$ of concepts with $C_i \not\equiv_{\mathcal{T}} C_j$ for $i \neq j$, the concepts

$$D_n := \prod_{i \leq n} C_i \sqcap D \quad \text{and} \quad E_n := \prod_{i \leq n} C_i \sqcap E$$

fulfill the condition $\lim_{n\to\infty} m_{\mathcal{T}}(D_n, E_n) = 1$. However, the measures we have introduced are oblivious to the commonalities the concepts have. In particular, to achieve this, we might have to sacrifice the property that measures are based on the symmetric difference of the underlying languages. As a result, it will no longer be guaranteed that all the other properties are satisfied. Even if they were, one would need to prove this again, under the new setting.

5.5 Computability of CDMs

Finally, we want to investigate computability of the functions measuring the distance between concepts. What we in principle need is, given a CDM m to derive a procedure that takes as input two concept descriptions C, D and a general TBox $\mathcal{T}$ and outputs $m_{\mathcal{T}}(C, D)$.

One of the main advantages of our framework is the modular way in which CDMs are defined. In particular, this provides us with a straightforward procedure of computing the distance between the input concept descriptions in the presence of a TBox, based on previous results on the constituting elements. More precisely, a CDM defined under our framework, i.e., induced from the language distance d and combining function f, operates as follows: it first derives the languages $\mathcal{L}_{\mathcal{T}}(C, A_i), \mathcal{L}_{\mathcal{T}}(D, A_i)$ for every $A_i \in \mathsf{N_C}$, then it computes $d(\mathcal{L}_{\mathcal{T}}(C, A_i), \mathcal{L}_{\mathcal{T}}(D, A_i))$, that is their distance under d, and finally combines these into a single value using the combining function f. It is hence apparent what we need in order to compute the final value:

1. a way to derive the language $\mathcal{L}_{\mathcal{T}}(C, A)$ given an $\mathcal{FL}_0$ TBox $\mathcal{T}$, a concept description C, and a concept name A,
2. a way to compute the distance of two languages, and
3. a computable combining function.

For (1), Theorem 4.6 provides us with an ExpTime procedure of deriving such a DFA. Computing language distances was the main focus of Section 3.5. The main idea was to combine an rLTA representing the input languages with a wLTA that corresponds to the distance function. By Lemma 3.21, we can combine the DFAs obtained by Theorem 4.6 for the input languages in quadratic time to derive such an rLTA. Computability of the distances d_0, d_1, and d_2 in polynomial time was a particular result of this procedure, as we saw in Theorem 3.45. Lastly, picking a computable combining function is an easy task. Overall, we obtain the following.

Theorem 5.5. *The CDM $m^{d,f}$ induced from a combining function f that is computable in polynomial time and a language distance d that can be expressed by a Φ-wLTA[2], is computable in* ExpTime*, that is, given an $\mathcal{FL}_0$ TBox $\mathcal{T}$ and concept descriptions C, D, we can compute $m^{d,f}_{\mathcal{T}}(C, D)$ in time exponential in the combined size of $\mathcal{T}, C, D$.*

[2]The Φ-wLTA has to actually satisfy the restrictions of Theorem 3.45.

Chapter 6

Approximate Unification in the Description Logic $\mathcal{FL}_0$

In this chapter we investigate a relaxed version of unification that we call *approximate*. As motivated in the introduction, given patterns C and D it makes sense to look for substitutions σ that are actually not unifiers, but come close to being unifiers, in the sense that the distance between $\sigma(C)$ and $\sigma(D)$ is small. We call such substitutions *approximate unifiers*.

Subsequently, we will first introduce approximate unification based on concept distance measures (CDMs) in the description logic $\mathcal{FL}_0$, and then we will proceed to solve this problem. Our techniques for approximate unification are based on the approach from [BN01] on classical unification that we introduced in Chapter 4. In particular, we will first introduce approximately solving language equations based on distances between languages. As we have seen in Chapter 5, language distances can be used to define CDMs. Based on this relation, we show that approximate unification for CDMs obtained this way can be reduced to approximately solving language equations. We conclude the chapter with the technical part of how to check approximate solvability of these equations.

6.1 Definition

We will first define the problem with respect to general TBoxes. However, our techniques for decidability only apply for the case of the empty TBox. Therefore, in the rest of this chapter we will only deal with this setting.

Definition 6.1 (Approximate unification). *Given a CDM m, $\mathcal{FL}_0$ concept patterns C, D, TBox $\mathcal{T}$ and a substitution σ, the* degree of violation *of σ is defined as*

$$\nu_{m_\mathcal{T}}(\sigma, C, D) := m_\mathcal{T}(\sigma(C), \sigma(D)).$$

*For $p \in \mathbb{Q}$, we say that σ is a p-*approximate unifier of C, D w.r.t. $\mathcal{T}$ *if* $2^{-p} > \nu_{m_\mathcal{T}}(\sigma, C, D)$.$\diamondsuit$

Equivalence closedness of m yields that $\nu_{m_\mathcal{T}}(\sigma, C, D) = 0$ iff σ is a unifier of C, D w.r.t. $\mathcal{T}$.

The *decision problem* for approximate unification asks, for a given threshold $p \in \mathbb{Q}$, whether C, D have a p-approximate unifier w.r.t. $\mathcal{T}$ or not.[1] In addition, we consider the following *computation problem*: compute $\inf_{\sigma \in Sub} \nu_{m_\mathcal{T}}(\sigma, C, D)$. The following lemma, which is immediate from the definitions, shows that a solution of the computation problem also yields a solution of the decision problem.

[1] The reason for formulating the problem like this will become clear in Section 6.3. In particular, our reduction can only prove hardness of the problem if the threshold is provided in this very concise form.

Lemma 6.2. *Let m be a concept distance, C, D $\mathcal{FL}_0$ concept patterns, and $\mathcal{T}$ TBox. Then C, D have a p-approximate unifier w.r.t. $\mathcal{T}$ iff $2^{-p} > \inf_{\sigma \in Sub} \nu_{m_\mathcal{T}}(\sigma, C, D)$.*

Proof. By definition, if C, D have a p-approximate unifier w.r.t. $\mathcal{T}$, then there exists a substitution $\sigma \in Sub$, s.t. $2^{-p} > \nu_{m_\mathcal{T}}(\sigma, C, D)$, and thus $2^{-p} > \inf_{\sigma \in Sub} \nu_{m_\mathcal{T}}(\sigma, C, D)$.

Suppose now that $2^{-p} > \inf_{\sigma \in Sub} \nu_{m_\mathcal{T}}(\sigma, C, D)$. By the definition of infimum, for every $\varepsilon > 0$, there exists a $\sigma_\varepsilon \in Sub$ s.t. $\nu_{m_\mathcal{T}}(\sigma_\varepsilon, C, D) \leqslant \inf_{\sigma \in Sub} \nu_{m_\mathcal{T}}(\sigma, C, D) + \varepsilon$. Thus, for $\varepsilon < 2^{-p} - \inf_{\sigma \in Sub} \nu_{m_\mathcal{T}}(\sigma, C, D)$ we have the required result. □

The reduction of the decision problem to the computation problem obtained from this lemma is actually polynomial. In fact, though the size of a representation of the number 2^{-p} may be exponential in the size of a representation of p, the number 2^{-p} need not be computed. Instead, we can compare p with $\log_2 \inf_{\sigma \in Sub} \nu_m(\sigma, C, D)$, where for the comparison we only need to compute as many digits of the logarithm as p has.

As we discussed in Chapter 4, even (exact) unification w.r.t. general TBoxes is not known to be decidable, and most results deal with the case of the empty TBox. Therefore, we will also restrict our focus to this case. Furthermore, we will simplify the notation: by a slight abuse of notation, we will write m to denote $m_\emptyset$, the particular function of the family m that corresponds to the empty TBox. Moreover, we will not refer to TBoxes whatsoever; the rest of the results in this chapter are with respect to the empty TBox.

Recall that with respect to the empty TBox, the finite languages obtained by the normal form (4.2) coincide with the value restriction sets, i.e., for an $\mathcal{FL}_0$ concept description C with normal form $C \equiv \forall K_1.A_1 \sqcap \cdots \sqcap \forall K_k.A_k$ it holds that $\mathcal{L}_\emptyset(C, A_i) = K_i$.

6.2 Reducing to language equations

Following the approach that was used for exact unification in $\mathcal{FL}_0$, we will utilize the close connection with formal languages. Recall that unification of two $\mathcal{FL}_0$ concept patterns reduces to solving language equations of the form (4.4), or equivalently of the form (4.6).

In this section we will demonstrate how approximate $\mathcal{FL}_0$ unification can be reduced to the following problem of *approximately solving language equations.* As in [BO13], instead of equations of the form (4.4) or (4.6), we consider the more general case of equations between expressions of the form (4.7) or (4.8).

Definition 6.3 (Approximate solvability of language equations). *Given a language distance d, language expressions ϕ, ψ, and an assignment σ, the* degree of violation *of σ is defined as $\nu_d(\sigma, \phi, \psi) := d(\sigma(\phi), \sigma(\psi))$. For $p \in \mathbb{Q}$, we say that σ is a p-*approximate solution of *$\phi \approx^? \psi$ if $2^{-p} > \nu_d(\sigma, \phi, \psi)$.* ♢

The *decision* and the *computation problem* for approximately solving language equations are defined analogously to the case of unification. In addition, the analog of Lemma 6.2 also holds in this case, and thus the decision problem can be reduced to the computation problem.

In this setting, equivalence of equations between expressions of the form (4.7) and of the form (4.8) does not come for free, as for exact case. It still holds, however, when we consider language distances d that are *invariant under mirroring*, i.e., for every pair of languages L, M it holds that $d(L, M) = d(L^{mi}.M^{mi})$. In fact, let d be such a distance and ϕ, ψ be both

expressions of one of the forms (4.7) or (4.8), and σ an assignment. By induction, we can prove that $\sigma(\phi)^{mi} = \sigma^{mi}(\phi^{mi})$, and thus

$$\begin{aligned} v_d(\sigma, \phi, \psi) &= d(\sigma(\phi), \sigma(\psi)) = d(\sigma(\phi)^{mi}, \sigma(\psi)^{mi}) \\ &= d(\sigma^{mi}(\phi^{mi}), \sigma^{mi}(\psi^{mi})) = v_d(\sigma^{mi}, \phi^{mi}, \psi^{mi}). \end{aligned}$$

Hence, for this class of distances, to which d_0, d_1 and d_2 belong in particular, we can consider equations between expressions of either form interchangably.

Recall that unification in $\mathcal{FL}_0$ is reduced to *finite* solvability of language equations. The above definition of approximately solving language equations and of the decision and the computation problem can also be restricted to finite assignments, in which case we talk about *finite approximate solvability*. However, we will show that finite approximate solvability can actually be reduced to (general) approximate solvability. For this to be the case, we need the language distance to satisfy an additional property $(M4)$. Given a natural number ℓ, we call two languages $K, L \subseteq \Sigma^*$ *equal up to length* ℓ (and write $K \equiv_\ell L$) if K and L coincide on all words of length at most ℓ.

> $(M4)$ Let L be a language and (L_n) a sequence of languages over Σ.
> Then, $L_n \equiv_n L$ for all $n \geq 0$ implies $L_n \xrightarrow{d} L$.[2]

Note that, in particular, $(M4)$ holds for d_1 and d_2: the assumption that $L_n \equiv L \mod \Sigma^{\leqslant n}$ implies that $d_1(L_n, L) \leqslant 2^{-(n+1)}$ and likewise

$$d_2(L_n, L) = \frac{1}{2} \sum_{w \in L_n \triangle L} (2|\Sigma|)^{-|w|} \leqslant \frac{1}{2} \sum_{w \in \Sigma^* \setminus \Sigma^{\leqslant n}} (2|\Sigma|)^{-|w|} = \frac{1}{2^{n+1}}.$$

Thus $L_n \xrightarrow{d_1} L$ and also $L_n \xrightarrow{d_2} L$.

Note however, that $(M4)$ does not hold for d_0. As a counterexample, consider any infinite language L, and set $L_n := L \cap \Sigma^{\leq n}$. Then it holds that $L_n \equiv_n L$ for all $n \geq 0$, but $d_0(L_n, L) = \infty$, and hence $L_n \not\xrightarrow{d} L$. Hence, the rest of the results in this section do not apply for d_0.

If $(M4)$ is satisfied for d, then the computation problem for finite assignments has the same solution as for arbitrary assignments.

Lemma 6.4. *Let d be a language distance satisfying $(M4)$ and ϕ, ψ language expressions. Then,*

$$\inf_{\sigma \in finAss} v_d(\sigma, \phi, \psi) = \inf_{\sigma \in Ass} v_d(\sigma, \phi, \psi).$$

Proof. Obviously,

$$\inf_{\sigma \in finAss} v_d(\sigma, \phi, \psi) \geqslant \inf_{\sigma \in Ass} v_d(\sigma, \phi, \psi).$$

Set $p = \inf_{\sigma \in Ass} v_d(\sigma, \phi, \psi)$. This means that there exists a sequence of assignments $\sigma_1, \sigma_2, \ldots$, s.t. $v_d(\sigma_n, \phi, \psi) \xrightarrow{d_\infty} p$. By $(M4)$, for each σ_i, there exists a sequence of finite assignments $\sigma_i^{(1)}, \sigma_i^{(2)}, \ldots$, s.t. $v_d(\sigma_i^{(n)}, \phi, \psi) \xrightarrow{d_\infty} v_d(\sigma_i, \phi, \psi)$. We will construct

[2] The definition of convergence w.r.t. a metric d can be found in the Appendix.

a sequence of finite assignments $\tau_1, \tau_2, \ldots$, s.t. $\nu_d(\sigma_i^{(n)}, \phi, \psi) \xrightarrow{d_\infty} p$. This implies that $\inf_{\sigma \in finAss} \nu_d(\sigma, \phi, \psi) \leqslant p$, and the proof is complete.

By definition of convergence, we have that for every $n \in \mathbb{N}$:

- there exists σ_{i_n} s.t. $d_\infty(\nu_d(\sigma_{i_n}, \phi, \psi), p) < \frac{1}{2n}$,
- there exists $\sigma_{i_n}^{j_n}$ s.t. $d_\infty(\nu_d(\sigma_{i_n}^{j_n}, \phi, \psi), \nu_d(\sigma_{i_n}, \phi, \psi)) < \frac{1}{2n}$.

Thus, by the triangle inequality, we get that $d_\infty(\nu_d(\sigma_{i_n}^{j_n}, \phi, \psi), p) < \frac{1}{n}$. Set $\tau_n = \sigma_{i_n}^{j_n}$ and we have the required sequence. □

To see that this result does not hold for d_0, consider the language equation $\{\varepsilon\} \cup \{a\}X =^? X$. For every finite assignment σ we have that $\nu_{d_0}(\sigma, \{\varepsilon\} \cup \{a\}X, X) \geq 1$, while the infinite assignment $\sigma' := \{X \mapsto a^*\}$ is actually a solution, and hence $\nu_{d_0}(\sigma', \{\varepsilon\} \cup \{a\}X, X) = 0$.

Reducing approximate unification to approximately solving language equations

In the following, we assume that d is a language distance, f a combining function, and $m^{d,f}$ the CDM induced by f, d. Let C, D be $\mathcal{FL}_0$ concept patterns in normal form, as shown in (4.3), and (4.4) the corresponding language equations, for $i = 1, \ldots, k$. We denote the left- and right-hand sides of the equations (4.4) with ϕ_i and ψ_i, respectively. The following lemma shows that the degree of violation transfers from finite assignments $\sigma_1, \ldots, \sigma_k$ to the induced substitution $\rho(\sigma_1, \ldots, \sigma_k)$ as defined in (4.5).

Lemma 6.5. *Let $\sigma_1, \ldots, \sigma_k$ be finite assignments. Then*

$$f(\nu_d(\sigma_1, \phi_1, \psi_1), \ldots, \nu_d(\sigma_k, \phi_k, \psi_k)) = \nu_{m^{d,f}}(\rho(\sigma_1, \ldots, \sigma_k), C, D).$$

Proof. Consider the concept patterns

$$C = \forall S_{0,1}.A_1 \sqcap \ldots \sqcap \forall S_{0,k}.A_k \sqcap \forall S_1.X_1 \sqcap \ldots \sqcap \forall S_n.X_n$$
$$D = \forall T_{0,1}.A_1 \sqcap \ldots \sqcap \forall T_{0,k}.A_k \sqcap \forall T_1.X_1 \sqcap \ldots \sqcap \forall T_n.X_n$$

Set $L_{i,j} := \sigma_j(X_i)$ for $i = 1, \ldots, n,\ j = 1, \ldots, k$. Recall that ρ is the bijection between tuples of assignments and substitutions described in 4.5, and in the following abbreviate $\rho(\sigma_1, \ldots, \sigma_k)$ by σ.

Then we have that

$$\sigma(C) = \prod_{i=1}^{k} \forall(S_{0,i} \cup S_1 L_{1,i} \cup \cdots \cup S_n L_{n,i})A_i$$
$$\sigma(D) = \prod_{i=1}^{k} \forall(T_{0,i} \cup T_1 L_{1,i} \cup \cdots \cup T_n L_{n,i})A_i$$

and

$$\sigma_i(\varphi_i) = S_{0,i} \cup S_1 L_{1,i} \cup \cdots \cup S_n L_{n,i}$$
$$\sigma_i(\psi_i) = T_{0,i} \cup T_1 L_{1,i} \cup \cdots \cup T_n L_{n,i}.$$

Thus

$$\begin{aligned}
\nu_{m^{d,f}}(\sigma, C, D) =& m^{d,f}(\sigma(C), \sigma(D)) \\
=& f(d(S_{0,1} \cup S_1 L_{1,1} \cup \cdots \cup S_n L_{n,1}, T_{0,1} \cup T_1 L_{1,1} \cup \cdots \cup T_n L_{n,1}), \\
& \ldots, d(S_{0,k} \cup S_1 L_{1,k} \cup \cdots \cup S_n L_{n,k}, T_{0,k} \cup T_1 L_{1,k} \cup \cdots \cup T_n L_{n,k})) \\
=& f(d(\sigma_1(\varphi_1), \sigma_1(\psi_1)), \ldots, d(\sigma_k(\varphi_k), \sigma_k(\psi_k))) \\
=& f(\nu_d(\sigma_1, \varphi_1, \psi_1), \ldots, \nu_d(\sigma_k, \varphi_k, \psi_k))
\end{aligned}$$ □

Since the combining function is continuous and monotone, by Lemma A.4 the equality stated in this lemma is preserved under building the infimum. In addition, Lemma 6.4 shows that the restriction to finite assignments can be dispensed with if d satisfies $(M4)$.

Lemma 6.6. *Assume that d satisfies $(M4)$. Then,*

$$\begin{aligned}
&\inf_{\sigma \in Sub} \nu_{m^{d,f}}(\sigma, C, D) = \\
&\qquad = f\left(\inf_{\sigma_1 \in finAss} \nu_d(\sigma_1, \phi_1, \psi_1), \ldots, \inf_{\sigma_k \in finAss} \nu_d(\sigma_k, \phi_k, \psi_k)\right) \\
&\qquad = f\left(\inf_{\sigma_1 \in Ass} \nu_d(\sigma_1, \phi_1, \psi_1), \ldots, \inf_{\sigma_k \in Ass} \nu_d(\sigma_k, \phi_k, \psi_k)\right).
\end{aligned}$$

In case f is computable (in polynomial time), this lemma yields a (polynomial time) reduction of the computation problem for approximate $\mathcal{FL}_0$ unification to the computation problem for approximately solving language equations. In addition, we know that the decision problem can be reduced to the computation problem. Thus, it is sufficient to devise a procedure for the computation problem for approximately solving language equations.

6.3 Approximately solving language equations

In the following, we show how to solve the computation problem for language distances that are defined using the symmetric difference of the input languages and can be expressed with weighted tree automata, as described in Section 3.5. In particular, d_0, d_1 and d_2 are such instances. Our solution adapts the automata-based approach for solving language equations introduced in [BO13], that we described in Section 4.4.

Similarly to the normalization step, given an approximate equation $\phi \approx^? \psi$, we first derive *two* normalized equations $\phi_s =^? \emptyset$ and $\phi_a \approx^? \emptyset$, the first to be solved strictly and the second to be solved approximately. More precisely, we have the following for language distances d defined using the symmetric difference of the input languages, i.e., if there exists a function $f : 2^{\Sigma^*} \to [0, +\infty]$ such that $d(K, L) = f(K \,\Delta\, L)$ for every $K, L \subseteq \Sigma^*$.

Lemma 6.7. *Let ϕ, ψ be language expressions. Then we can compute in polynomial time normalized language expressions ϕ_a and ϕ_s such that the following holds for every d defined using the symmetric difference of the input languages:*

$$\{\nu_d(\sigma, \phi, \psi) \mid \sigma \in Ass\} = \{\nu_d(\sigma, \phi_a, \emptyset) \mid \sigma \in Ass \wedge \sigma(\phi_s) = \emptyset\}.$$

Proof. In [BO13] (Lemma 1) it is shown how a given system of language equations can be transformed into a single normalized language equation such that there is a one-to-one correspondence between the solutions of the original system and the solutions of the normal form. Given an approximate equation $\phi \approx^? \psi$, we first abstract the left- and right-hand side of this equation with new variables X, Y, and add strict equations that say that X must be equal to ϕ and Y must be equal to ψ, i.e., we consider the approximate equation $X \approx^? Y$ together with the strict equations $X =^? \phi$ and $Y =^? \psi$. We then apply the normalization approach of [BO13] to the two strict equations, to obtain the normalized strict equation $\phi_s =^? \emptyset$. Though it is not explicitely stated in [BO13], it is easy to see that this transformation is such that, for any assignment σ of the original equation, there is a solution θ of $\phi_s =^? \emptyset$ such that $\theta(X) = \sigma(\phi)$ and $\theta(Y) = \sigma(\psi)$. Conversely, any solution θ of $\phi_s =^? \emptyset$ satisfies $\theta(X) = \sigma(\phi)$ and $\theta(Y) = \sigma(\psi)$ for some assignment σ of the original equation. Consequently, we have

$$\{(\sigma(\phi), \sigma(\psi)) \mid \sigma \in Ass\} = \{(\theta(X), \theta(Y)) \mid \theta \in Ass \wedge \theta(\phi_s) = \emptyset\}.$$

If we now define $\phi_a := X \triangle Y$, then the lemma is an easy consequence of the above identity and the fact that d considers the symmetric difference of the input languages, and thus $d(K, L) = f(K \triangle L) = d(K \triangle L, \emptyset)$. □

This lemma shows that, to solve the computation problem for $\phi \approx^? \psi$, we can solve the computation problem for $\phi_a \approx^? \emptyset$, but restrict the infimum to assignments that solve the strict equation $\phi_s =^? \emptyset$, that is,

$$\inf_{\sigma \in Ass} \nu_d(\sigma, \phi, \psi) = \inf_{\substack{\sigma \in Ass \\ \sigma(\phi_s) = \emptyset}} \nu_d(\sigma, \phi_a, \emptyset) = \inf_{\substack{\sigma \in Ass \\ \sigma(\phi_s) = \emptyset}} d(\sigma(\phi_a), \emptyset). \tag{6.1}$$

Next, recall that [BO13] would construct an LTA corresponding to the normalized equation. We adapt this approach as follows: we apply this construction to the expression $\phi := \phi_a \cup \phi_s$, to obtain an LTA $\mathcal{A}^\phi = (\Sigma, Q^\phi, \{\ell\}, \Delta^\phi, I^\phi)$ with designated sets of states $F_1, \ldots, F_m \subseteq Q^\phi$, as described in Proposition 4.12.

Recall that every run r of $\mathcal{A}^\phi$ induces an assignment σ_r and every assignment is induced by some run of $\mathcal{A}^\phi$. By Proposition 4.12, we obtain sets of states $F_s := F_{\phi_s}$ and $F_a := F_{\phi_a}$ such that $\sigma_r(\phi_s) = L_r(\mathcal{A}^\phi, F_s)$ and $\sigma_r(\phi_a) = L_r(\mathcal{A}^\phi, F_a)$. holds for all runs r of A^ϕ.

By removing all states of F_s from A^ϕ, we obtain an automaton whose runs are in one-to-one correspondence with the assignments that solve $\phi_s =^? \emptyset$. In addition, we can make this automaton *trim*[3] using the polytime construction in the proof of Lemma 2 in [BO13].

Theorem 6.8. *Given an approximate equation $\phi_a \approx^? \emptyset$ and a strict equation $\phi_s =^? \emptyset$, we can construct in exponential time a trim LTA $\mathcal{A} = (\Sigma, Q, \{\ell\}, \Delta, I)$ and sets of states $F_a, F_1, \ldots, F_m \subseteq Q$ such that every run r of $\mathcal{A}$ satisfies $\sigma_r(\phi_a) = L_r(\mathcal{A}, F_a)$ and $\sigma_r(\phi_s) = \emptyset$. In addition, every assignment σ with $\sigma(\phi_s) = \emptyset$ is induced by some run of $\mathcal{A}$.*

Hence we obtain that

$$\inf_{\substack{\sigma \in Ass \\ \sigma(\phi_s) = \emptyset}} d(\sigma(\phi_a), \emptyset) = \inf_{r \in R_\mathcal{A}} d(\sigma_r(\phi_a), \emptyset) \tag{6.2}$$

[3] An LTA $\mathcal{A} = (\Sigma, Q, \{\ell\}, \Delta, I)$ is *trim* if every state can be used by some run of the LTA.

Now assume that we are given a Φ-wLTA $\mathcal{M} = (\Sigma, P, \{0,1\}, in, wt)$ that operates on $\{0,1\}$-labeled trees and expresses the language distance d, i.e., for any pair of languages K, M over Σ it holds that $d(K,M) = (||\mathcal{M}||, t_{K \Delta M})$.

First, we derive an "equivalent" Φ-wLTA that operates on Q-labeled trees as follows. Define the mapping $\theta : Q \to \{0,1\}$ by setting $\theta(q) = 1$ iff $q \in F_a$, and extend it to a mapping from Q-labeled trees to $\{0,1\}$-labeled trees by setting $\theta(t)(w) = \theta(t(w))$ for every $t \in T^{\omega}_{\Sigma,Q}$ and $w \in \Sigma^*$. Note that for every run r of $\mathcal{A}$ (on the unlabeled tree), since r itself is a Q-labeled tree, Theorem 6.8 together this construction gives us that

$$\theta(r) = t_{\sigma_r(\phi_a)}. \tag{6.3}$$

Next, construct the Φ-wLTA $\mathcal{M}' = (\Sigma, P, Q, in, wt')$ by setting for every $(p_0, p_1, \ldots, p_k) \in P^{k+1}$ and $q \in Q$ that $wt'(p_0, q, p_1, \ldots, p_k) = wt(p_0, \theta(q), p_1, \ldots, p_k)$. The next lemma formulates the exact meaning of the claim that $\mathcal{M}'$ is equivalent to $\mathcal{M}$.

Lemma 6.9. *For every tree $t \in T^{\omega}_{\Sigma,Q}$ it holds that $(||\mathcal{M}'||, t) = (||\mathcal{M}||, \theta(t))$.*

To prove this claim, note that every run of $\mathcal{M}'$ on t can also be seen as a run of $\mathcal{M}$ on $\theta(t)$ and vice versa. Thus we get

$$\begin{aligned}
(||\mathcal{M}||, \theta(t)) &= \inf_{r \in R_{\mathcal{M}}(\theta(t))} \left\{ in(r(\varepsilon)) + \sum_{w \in \Sigma^*} wt(r, w) \right\} \\
&= \inf_{r \in R_{\mathcal{M}}(\theta(t))} \left\{ in(r(\varepsilon)) + \sum_{w \in \Sigma^*} wt(r(w), \theta(t(w)), r(w\sigma_1), \ldots, r(w\sigma_k)) \right\} \\
&= \inf_{r \in R_{\mathcal{M}'}(t)} \left\{ in(r(\varepsilon)) + \sum_{w \in \Sigma^*} wt'(r(w), t(w), r(w\sigma_1), \ldots, r(w\sigma_k)) \right\} \\
&= (||\mathcal{M}'||, t).
\end{aligned}$$

Combining all the above, we have that

$$\begin{aligned}
\inf_{\sigma \in Ass} v_d(\sigma, \phi, \psi) &\overset{(6.1)}{=} \inf_{\substack{\sigma \in Ass \\ \sigma(\phi_s) = \emptyset}} d(\sigma(\phi_a), \emptyset) \overset{(6.2)}{=} \inf_{r \in R_{\mathcal{A}}} d(\sigma_r(\phi_a), \emptyset) \\
&\overset{(*)}{=} \inf_{r \in R_{\mathcal{A}}} (||\mathcal{M}||, t_{\sigma_r(\phi_a)}) \overset{(6.3)}{=} \inf_{r \in R_{\mathcal{A}}} (||\mathcal{M}||, \theta(r)) \\
&\overset{\text{Lemma 6.9}}{=} \inf_{r \in R_{\mathcal{A}}} (||\mathcal{M}'||, r),
\end{aligned}$$

where equality $(*)$ holds because $\mathcal{M}$ expresses the language distance d.

Hence, in order to solve our initial computation problem it suffices to be able to compute $\inf_{r \in R_{\mathcal{A}}} (||\mathcal{M}'||, r)$. To this end, if $\mathcal{M}$ is over $\mathbb{R}_{\inf}$ we can adapt the approach we followed in Section 3.6: we combine the wLTA with the LTA into a new wLTA that operates on the unlabeled tree. More precisely, we have the following lemma.

Lemma 6.10. *Given a Φ-wLTA $\mathcal{M}$ over $\mathbb{R}_{\inf}$ operating on Q-labeled trees and an LTA $\mathcal{A}$ with state set Q, one can construct in polynomial time a Φ-wLTA $\mathcal{M}_{\mathcal{A}}$ such that*

$$\inf_{r \in R_{\mathcal{A}}} (||\mathcal{M}||, r) = (||\mathcal{M}_{\mathcal{A}}||, t_{ul}).$$

Proof. Assume that $\mathcal{A} = (\Sigma, Q, \{\ell\}, \Delta, I)$ with sets of designated sets of states $F_1, \ldots, F_m$ and $\mathcal{M} = (\Sigma, P, Q, in, wt)$ over $\mathbb{R}_{\text{inf}}$. We define the Φ-wLTA $\mathcal{M}_{\mathcal{A}} = (\Sigma, P \times Q, in', wt')$ over $\mathbb{R}_{\text{inf}}$ as follows:

$$in'(p,q) := \begin{cases} in(p) & \text{if } q \in I \\ \infty & \text{otherwise} \end{cases}$$

$$wt'\big((p_0,q_0),(p_1,q_1),\ldots,(p_k,q_k)\big) := \begin{cases} wt(p_0,q_0,p_1,\ldots,p_k) & \text{if } (q_0,q_1,\ldots,q_k) \in \Delta, \\ \infty & \text{otherwise} \end{cases}$$

To prove that $\inf_{r \in R_{\mathcal{A}}}(||\mathcal{M}||, r) = (||\mathcal{M}_{\mathcal{A}}||, t_{ul})$, it is sufficient to show that there exists an injection $\tau : \bigcup_{r \in R_{\mathcal{A}}} R_{\mathcal{M}}(r) \to R_{\mathcal{M}_{\mathcal{A}}}$ such that for every $r \in R_{\mathcal{A}}$ and every $s \in R_{\mathcal{M}}(r)$ it holds that $weight(s) = weight(\tau(s))$, and $weight(s') = \infty$ for every $s' \in R_{\mathcal{M}_{\mathcal{A}}} \setminus im(\tau)$, where $im(\tau)$ stands for the *image* set of the mapping τ.

Given a run $r \in R_{\mathcal{A}}$, for every $s \in R_{\mathcal{M}}(r)$ set $\tau(s) = s'$ such that $s'(w) = (s(w), r(w))$. Then, for every $s \in R_{\mathcal{M}}(r)$ and every $w \in \Sigma^*$ we have that

$$\begin{aligned} wt(s', w) &= wt'(\overrightarrow{s'}(w)) = wt'(s'(w), s'(w\sigma_1), \ldots, s'(w\sigma_k)) \\ &= wt'((s(w), r(w)), (s(w\sigma_1), r(w\sigma_1)), \ldots, (s(w\sigma_k), r(w\sigma_k))) \\ &\overset{(*)}{=} wt(s(w), r(w), s(w\sigma_1), \ldots, s(w\sigma_k)) \\ &= wt(\overrightarrow{s}(w)) = wt(s, w), \end{aligned}$$

where equality $(*)$ holds since $(r(w), r(w\sigma_1), \ldots, r(w\sigma_k)) \in \Delta$.

Thus we obtain

$$\begin{aligned} weight(s') &= in(s'(\varepsilon)) + \sum_{w \in \Sigma^*} \phi_w(wt(s', w)) \\ &= in(s(\varepsilon)) + \sum_{w \in \Sigma^*} \phi_w(wt(s, w)) = weight(s). \end{aligned}$$

Now, suppose that $s' \in R_{\mathcal{M}_{\mathcal{A}}} \setminus im(\tau)$. Assume that $s'(w) = (s_0(w), r_0(w))$. If r_0 is a run of $\mathcal{A}$, since s_0 (like any P-labeled tree) can be seen as a run of $\mathcal{M}$ on r_0, i.e., $s_0 \in R_{\mathcal{M}}(r_0)$, and hence we have that $s' = \tau(s_0)$, a contradiction. Thus we have that $r_0 \notin R_{\mathcal{A}}$. We distinguish two cases.

- $r_0(\varepsilon) \notin I$. Then $in'(s'(\varepsilon)) = \infty$, and thus $weight(s') = \infty$.
- There exists $v \in \Sigma^*$ such that $(r_0(v), r_0(v\sigma_1), \ldots, r_0(v\sigma_k)) \notin \Delta$. Then

  $$wt(s', v) = wt'\big((s_0(v), r_0(v)), (s_0(v\sigma_1), r_0(v\sigma_1)), \ldots, (s_0(v\sigma_k), r_0(v\sigma_k))\big) = \infty,$$

  and thus $weight(s') = \infty$.

Finally, we get that

$$\begin{aligned}(||\mathcal{M}_{\mathcal{A}}||, t_{ul}) &= \inf_{s' \in R_{\mathcal{M}_{\mathcal{A}}}} weight(s') = \inf_{s' \in im(\tau)} weight(s') \\ &= \inf_{s \in \bigcup_{r \in R_{\mathcal{A}}} R_{\mathcal{M}}(r)} weight(s) = \inf_{r \in R_{\mathcal{A}}} \inf_{s \in R_{\mathcal{M}}(r)} weight(s) \\ &= \inf_{r \in R_{\mathcal{A}}} (||\mathcal{M}||, r)\end{aligned}$$

□

For computing the behavior of the wLTA we obtained, which is also the answer to the original computation problem, we can apply the results of Section 3.7. Overall, we have the following result.

Theorem 6.11. *Let d be a language distance defined using the symmetric distance of the input languages that can be expressed by a Φ-wLTA over $\mathbb{R}_{\text{inf}}$ with nondecreasing or contracting discounting. The computational problem (and hence also the decision problem) of approximate solvability of language equations w.r.t. d can be answered in time exponential in the size of the problem.*

Furthermore, if d is invariant under mirroring and satisfies property $(M4)$ and the combining function f is computable in polynomial time, then the computational problem (and hence also the decision problem) of approximate $\mathcal{FL}_0$ unification w.r.t. $m^{d,f}$ can be answered in time exponential in the size of the problem.

The second statement follows immediately from the first due to Lemma 6.6.

Next, we examine in more detail the cases of the distances d_1 and d_2. In particular, we will provide specialized procedures for solving the computational problem, and through these constructions we will be able to also prove hardness of the corresponding decision problems.

The measure d_1

Using Lemma 6.7, Theorem 6.8, and the definition of d_1, it is easy to see that the computation problem for an approximate language equation $\phi \approx^? \psi$ can be reduced to solving the following problem for the trim LTA $\mathcal{A} = (\Sigma, Q, \{\ell\}, \Delta, I)$ of Theorem 6.8: compute $\sup_{r \in R_{\mathcal{A}}} \min\{|w| \mid r(w) \in F_a\}$. More formally, we have the following lemma.

Lemma 6.12. *If $\ell = \sup_{r \in R_{\mathcal{A}}} \min\{|w| \mid r(w) \in F_a\}$ then $\inf_{\sigma \in Ass} v_{d_1}(\sigma, \phi, \psi) = 2^{-\ell}$.*

In order to compute this supremum, it is sufficient to compute, for every state $q \in Q$, the *length $lpr(q)$ of the longest partial run of $\mathcal{A}$ starting with q that does not have states of F_a at non-leaf nodes*. More formally, we define:

Definition 6.13. *Recall that $\Sigma^{\leqslant \ell}$ denotes the set of all words over Σ of length at most ℓ. Given a trim LTA $\mathcal{A} = (\Sigma, Q, \{\ell\}, \Delta, I)$, a* partial run of $\mathcal{A}$ of length ℓ from a state $q \in Q$ *is a mapping $p : \Sigma^{\leqslant \ell} \to Q$ such that $p(\varepsilon) = q$ and $(p(w), p(w\sigma_1), \ldots, p(w\sigma_k)) \in \Delta$ for all $w \in \Sigma^{\leqslant \ell - 1}$ and $a \in \Sigma$. The* leaves *of p are the words of length ℓ. Furthermore, we say that a partial run of $\mathcal{A}$ of length ℓ avoids the set $A \subseteq Q$ if $p : \Sigma^{\leqslant \ell} \to Q \setminus A$. Finally, for every $q \in Q$ we have that*

$$lpr(q) := \sup\{\ell \mid \textit{there exists a partial run of } \mathcal{A} \textit{ of length } \ell \textit{ from the state } q \textit{ that avoids } F_a\}.$$

♢

Lemma 6.14. *The function $lpr : Q \to \mathbb{N} \cup \{\infty\}$ can be computed in time polynomial in the size of $\mathcal{A}$.*

Proof. In order to compute *lpr*, we use an iteration similar to the emptiness test for looping tree automata [BT01].

If $q \in F_a$, then clearly $lpr(q) = 0$ and otherwise q has an appropriate partial run of length greater than 0 (recall that $\mathcal{A}$ is trim). For this reason, we start the iteration with

$$Q^{(0)} := F_a.$$

Next, for $i \geq 0$, we define

$$Q^{(i+1)} := Q^{(i)} \cup \{q \mid \forall (q, q_1, \ldots, q_k) \in \Delta . \exists j . q_j \in Q^{(i)}\}.$$

We have $Q^{(0)} \subseteq Q^{(1)} \subseteq Q^{(2)} \subseteq \ldots \subseteq Q$. Since Q is finite, there is an index $j \leq |Q|$ such that $Q^{(j)} = Q^{(j+1)}$, and thus the iteration becomes stable.

It is easy to show that

$$lpr(q) = \begin{cases} \min\{i \mid q \in Q^{(i)}\} & \text{if } q \in Q^{(j)} \\ \infty & \text{if } q \notin Q^{(j)} \end{cases}$$

To prove the above, the following claim is enough.

Claim 6.15. *It holds that $q \notin Q^{(i)}$ iff there is a partial run of length $i+1$ of $\mathcal{A}$ starting with q that does not have states of F_a at non-leaf nodes.*

Proof (Claim). By induction on i:
For $i = 0$, $q \notin Q^{(i)}$ iff $q \notin F_a$ iff there is a partial run of length 1 of $\mathcal{A}$ starting with q that does not have states of F_a at non-leaf nodes (i.e. at the root).
For $i \geqslant 1$, if $q \notin Q^{(i)}$ then there exists $(q, q_1, \ldots, q_k) \in \Delta$ such that $q_1, \ldots, q_k \notin Q^{(i-1)}$. By the induction hypothesis, for every such q_j there is a partial run of length i of $\mathcal{A}$ starting with q_j that does not have states of F_a at non-leaf nodes, thus we can construct such a run of length $i+1$ for q. If $q \in Q^{(i)}$, then for every $(q, q_1, \ldots, q_k) \in \Delta$ it holds that there exists some j such that $q_j \in Q^{(i-1)}$. By the induction hypothesis, there is no partial run of length i of $\mathcal{A}$ starting with q_j that does not have states of F_a at non-leaf nodes, and thus $(q, q_1, \ldots, q_k)$ cannot be used to build a partial run of lenth $i+1$ starting with q. Since there exists such a q_j for every $(q, q_1, \ldots, q_k) \in \Delta$, this completes the proof of the claim.

If $q \notin Q^{(j)}$, note that the claim implies that there are such runs for every $n \in \mathbb{N}$, and thus $lpr(q) = \infty$.

Since the number of iterations is linear in $|Q|$ and every iteration step can obviously be performed in polynomial time, this completes the proof. □

The function *lpr* can now be used to solve the computation problem as follows:

$$\sup_{r \in R_{\mathcal{A}}} \min\{|w| \mid r(w) \in F_a\} = \max\{lpr(q) \mid q \in I\}.$$

If this maximum is ∞, then the measure d_1 yields value 0 and the approximate equation was actually solvable as a strict one.

Theorem 6.16. *For the distance d_1 and a polytime computable combining function, the computation problem (for approximate $\mathcal{FL}_0$ unification and for approximately solving language equations) can be solved in exponential time, and the decision problem is* ExpTime*-complete.*

Proof. The ExpTime-upper bounds follow from our reductions and the fact that the automaton $\mathcal{A}$ can be computed in exponential time and is thus of at most exponential size. Hardness can be shown by a reduction of the strict problems, which are known to be ExpTime-complete [BN01; BO13]. In fact, the proof of Lemma 6.14 shows that d_1 either yields the value $0 = 2^{-\infty}$ (in which case the strict equation is solvable) or a value larger than $2^{-(|Q|+1)}$ (in which case the strict equation is not solvable). In other words, for a threshold smaller than $2^{-(|Q|+1)}$ the decision problem is equivalent to the classical solvability problem. □

The measure d_2

Recall that the value of d_2 is obtained by applying the function μ to the symmetric difference of the input languages. In case one of the two languages is empty, its value is thus obtained by applying μ to the other language. It is easy to show that the following lemma holds.

Lemma 6.17. *The value $\mu(L)$ for $L \subseteq \Sigma^*$ satisfies the recursive equation:*

$$\mu(L) = \frac{1}{2}\chi_L(\varepsilon) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\mu(a^{-1}L), \tag{6.4}$$

where $a^{-1}L := \{w \in \Sigma^* \mid aw \in L\}$ and χ_L is the characteristic function of the language L, that takes the value 1 if $w \in L$ and 0 otherwise.

Proof. With simple calculations we get

$$\begin{aligned}
\mu(L) &= \frac{1}{2}\sum_{w\in L}(2|\Sigma|)^{-|w|} = \frac{1}{2}(2|\Sigma|)^{-|\varepsilon|}\chi_L(\varepsilon) + \frac{1}{2}\sum_{w\in L\setminus\{\varepsilon\}}(2|\Sigma|)^{-|w|} \\
&= \frac{1}{2}\chi_L(\varepsilon) + \frac{1}{2}\sum_{a\in\Sigma}\sum_{aw\in L}(2|\Sigma|)^{-(1+|w|)} = \frac{1}{2}\chi_L(\varepsilon) + \frac{1}{2}\sum_{a\in\Sigma}\sum_{w\in a^{-1}L}(2|\Sigma|)^{-1}(2|\Sigma|)^{-|w|} \\
&= \frac{1}{2}\chi_L(\varepsilon) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\frac{1}{2}\sum_{w\in a^{-1}L}(2|\Sigma|)^{-|w|} = \frac{1}{2}\chi_L(\varepsilon) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\mu(a^{-1}L).
\end{aligned}$$

□

Using Lemma 6.7, Theorem 6.8, and the definition of d_2, it is easy to see that the computation problem for an approximate language equation $\phi \approx^? \psi$ w.r.t. d_2 can be reduced to solving the following problem for the trim LTA $\mathcal{A} = (\Sigma, Q, \{\ell\}, \Delta, I)$ of Theorem 6.8: compute $\inf_{r\in R_\mathcal{A}} \mu(L_r(\mathcal{A}, F_a))$. In fact, this is the exact value that answers the computation problem, as by combining (6.1) and 6.2, we obtain that

$$\inf_{r\in R_\mathcal{A}} \mu(L_r(\mathcal{A}, F_a)) = \inf_{\sigma\in Ass} \nu_{d_2}(\sigma, \phi, \psi).$$

In the following, we are only interested in languages defined by runs of $\mathcal{A}$ with set of final states F_a, thus for ease of notation we will write L_r instead of $L_r(\mathcal{A}, F_a)$.

In order to obtain the technical results of this section, we need to look more closely to the LTA from Proposition 4.12 and their exact construction in [BO13]. In particular, these

are special instances of LTAs, that the authors of [BO13] call *looping tree automata with independent transitions (ILTA)*, since the state in each successor of a node is determined independently of the choice of the states in its siblings. In particular, their transition relation Δ is defined by using a transition function $\delta : Q \times \Sigma \to 2^Q$ in the fashion of an NFA; more precisely,

$$\Delta = \{(q, q_1, \ldots, q_k) \mid q \in Q, q_1 \in \delta(q, a_1), \ldots, q_k \in \delta(q, a_k)\}.$$

Using (6.4), we now show that this infimum can be computed by solving a system of recursive equations that is induced by the transitions of $\mathcal{A}$. Given an arbitrary (not necessarily initial) state $q \in Q$, we say that $r : \Sigma^* \to Q$ is a *q-run of $\mathcal{A}$* if $r(\varepsilon) = q$ and $r(wa) \in \delta(r(w), a)$ for all $w \in \Sigma^*$ and $a \in \Sigma$. We denote the set of all q-runs of $\mathcal{A}$ with $R^q_{\mathcal{A}}$. Since each run of $\mathcal{A}$ is a q_0-run for some $q_0 \in I$, we have

$$\inf_{r \in R_{\mathcal{A}}} \mu(L_r) = \min_{q_0 \in I} \inf_{r \in R^{q_0}_{\mathcal{A}}} \mu(L_r).$$

For all $q \in Q$, we define $\mu(q) := \inf_{r \in R^q_{\mathcal{A}}} \mu(L_r)$. The identity above shows that we can solve the computation problem for approximate language equations w.r.t. d_2 if we can devise a procedure for computing the values $\mu(q) \in \mathbb{R}$ for all $q \in Q$. The identity (6.4) can now be used to show the following lemma.

Lemma 6.18. *For all states $q \in Q$ we have*

$$\mu(q) = \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|} \sum_{a \in \Sigma} \min_{p \in \delta(q,a)} \mu(p),$$

where χ_{F_a} denotes the characteristic function of the set F_a.

Proof. Given a run $r \in R^q_{\mathcal{A}}$, for every $a \in \Sigma$ a unique run $r_a \in R^{r(a)}_{\mathcal{A}}$ is defined as $r_a(w) = r(aw)$. Obviously, $r(a) \in \delta(q, a)$. Conversely, given a run $r_a \in R^{q_a}_{\mathcal{A}}$, for every $a \in \Sigma$, such that $q_a \in \delta(q, a)$, a unique run $r_0 \in R^q_{\mathcal{A}}$ can be derived, as

$$r_0(w) = \begin{cases} q & \text{if } w = \varepsilon \\ r_a(u) & \text{if } w = au \end{cases}$$

Hence, there is a bijection between the set of q-runs $R^q_{\mathcal{A}}$ and the set of "successor" runs $SR(q) := \bigcup_{p_1 \in \delta(q,a_1)} R^{p_1}_{\mathcal{A}} \times \cdots \times \bigcup_{p_k \in \delta(q,a_k)} R^{p_k}_{\mathcal{A}}$.

Given a run $r \in R(q)$, it holds that

$$L_r = \varepsilon(q) \cup \bigcup_{a \in \Sigma} aL_{r_a}$$

where $\varepsilon(q) = \{\varepsilon\}$ if $q \in F_a$ and $\emptyset$ otherwise.

For the measure μ and disjoint sets of words $\mathcal{A}$ and B it holds that $\mu(A \dot\cup B) = \mu(A) + \mu(B)$. Additionally, for $a \in \Sigma$ and $A \subseteq \Sigma^*$, it holds $\mu(aA) = \frac{1}{2|\Sigma|}\mu(A)$.

Thus, given a run $r \in R^q_{\mathcal{A}}$ it holds

$$\begin{aligned}\mu(L_r) &= \mu(\varepsilon(q) \cup \bigcup_{a\in\Sigma} aL_{r_a}) = \mu(\varepsilon(q)) + \sum_{a\in\Sigma} \mu(aL_{r_a}) \\ &= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\mu(L_{r_a}).\end{aligned}$$

Thus it can be inferred that

$$\begin{aligned}\mu(q) &= \inf_{r\in R^q_{\mathcal{A}}} \mu(L_{r_q}) = \inf_{r\in R^q_{\mathcal{A}}} \left(\mu(\varepsilon\cdot\chi_{F_a}(q)) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\mu(L_{r_a})\right) \\ &= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\inf_{r\in R^q_{\mathcal{A}}}\sum_{i=1}^{k}\mu(L_{r_{a_i}}) = \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\inf_{(r_{a_1},\ldots,r_{a_k})\in SR(q)}\sum_{i=1}^{k}\mu(L_{r_{a_i}}) \\ &= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\min_{\substack{(p^1,\ldots,p^k)\in \\ \delta(q,a_1)\times\cdots\times\delta(q,a_k)}}\inf_{\substack{(r_1,\ldots,r_k)\in \\ R(p^1)\times\cdots\times R(p^k)}}\sum_{i=1}^{k}\mu(L_{r_i}) \\ &= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\min_{\substack{(p^1,\ldots,p^k)\in \\ \delta(q,a_1)\times\cdots\times\delta(q,a_k)}}\sum_{i=1}^{k}\inf_{r\in R(p^i)}\mu(L_r) \\ &= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{i=1}^{k}\min_{p\in\delta(q,a_i)}\inf_{r\in R(p)}\mu(L_r) = \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\min_{p\in\delta(q,a)}\mu(p). \qquad \square\end{aligned}$$

By introducing variables x_q (for $q \in Q$) that range over $\mathbb{R}$, we can rephrase this lemma by saying that the values $\mu(q)$ yield a solution to the system of equations

$$x_q = \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\min_{p\in\delta(q,a)} x_p \quad (q \in Q). \tag{6.5}$$

Thus, to compute the values $\mu(q)$ for $q \in Q$ it is sufficient to compute a solution of (6.5).

Next, we use Banach's fixed point theorem to show that the system has a *unique* solution in $\mathbb{R}$. In particular, we will transform the system of equations to a contraction in $\mathbb{R}^k$ that has as a fixed point the solution of (6.5).

For every equation (6.5) corresponding to a state $q_i \in Q$, we represent the right-hand side as a function $f_i : \mathbb{R}^k \to \mathbb{R}$, defined by

$$f_i(x_1,\ldots,x_k) = \frac{1}{2}\chi_{F_a}(q_i) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\min_{q_j\in\delta(q_i,a)} x_j$$

Next, the right-hand sides of the entire system are denoted by a vector function $f : \mathbb{R}^k \to \mathbb{R}^k$, with

$$f(x_1,\ldots,x_k) = (f_1(x_1,\ldots,x_k),\ldots,f_k(x_1,\ldots,x_k))$$

Proposition 6.19. *A vector* $\mathbf{x} = (x_1,\ldots,x_k)$ *is a solution of the system of equations of the form* (6.5) *iff it is a fixed point of* f. $\diamondsuit$

Before proving that f is a contraction, we provide a technical lemma that will be useful in the proof of the next one.

Lemma 6.20. *Given a finite set of indices I and a set $\{J(i) \subseteq I \mid i \in I\}$, it holds that*

$$\max_{i\in I}\left|\min_{j\in J(i)} x_j - \min_{j\in J(i)} y_j\right| \leqslant \max_{i\in I}|x_i - y_i|\,.$$

Proof. For every $i \in I$ we have

$$\min_{j\in J(i)} x_j = x_{k_i} \text{ for some } k_i \in J(i)$$
$$\min_{j\in J(i)} y_j = y_{\ell_i} \text{ for some } \ell_i \in J(i).$$

Then, for every $i \in I$ we get

$$\begin{aligned}\left|\min_{j\in J(i)} x_j - \min_{j\in J(i)} y_j\right| &= \left|x_{k_i} - y_{\ell_i}\right| \overset{(*)}{=} x_{k_i} - y_{\ell_i}\\ &\leqslant x_{\ell_i} - y_{\ell_i} \leqslant \max_{i\in I}|x_i - y_i|.\end{aligned}$$

For equality $(*)$ suppose without loss of generality, $x_{k_i} \geqslant y_{\ell_i}$. If $x_{k_i} \leqslant y_{\ell_i}$, the exact symmetric argument can be used.

Finally, we have that

$$\max_{i\in I}\left|\min_{j\in J(i)} x_j - \min_{j\in J(i)} y_j\right| \leqslant \max_{i\in I}\max_{i\in I}|x_i - y_i| = \max_{i\in I}|x_i - y_i|. \qquad \square$$

The following lemma provides the last condition for Theorem A.2.

Lemma 6.21. *The function f defined above is a contraction in $(\mathbb{R}^k, d_\infty)$.*

Proof. Let $\mathbf{x} = (x_1, \ldots, x_k)$, $\mathbf{y} = (y_1, \ldots, y_k) \in \mathbb{R}^k$. Then

$$\begin{aligned}d_\infty(f(\mathbf{x}),f(\mathbf{y})) &= \max_{i=1,\ldots,k}|f_i(\mathbf{x}) - f_i(\mathbf{y})|\\ &= \max_{i=1,\ldots,k}\left|\frac{1}{2}\chi_{F_a}(q_i) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\min_{q_j\in\delta(q_i,a)} x_j - \left(\frac{1}{2}\chi_{F_a}(q_i) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\min_{q_j\in\delta(q_i,a)} y_j\right)\right|\\ &= \max_{i=1,\ldots,k}\left|\frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\left(\min_{q_j\in\delta(q_i,a)} x_j - \min_{q_j\in\delta(q_i,a)} y_j\right)\right|\\ &\leqslant \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\max_{i=1,\ldots,k}\left|\left(\min_{q_j\in\delta(q_i,a)} x_j - \min_{q_j\in\delta(q_i,a)} y_j\right)\right|\\ &\overset{(*)}{\leqslant} \frac{1}{2|\Sigma|}\sum_{a\in\Sigma}\max_{i=1,\ldots,k}|x_i - y_i| = \frac{1}{2|\Sigma|}|\Sigma|\max_{i=1,\ldots,k}|x_i - y_i|\\ &= \frac{1}{2}\max_{i=1,\ldots,k}|x_i - y_i|\end{aligned}$$

where inequality $(*)$ holds because of Lemma 6.20. $\square$

Finally, since $(\mathbb{R}^k, d_\infty)$ is complete, from Theorem A.2 and Proposition 6.19 we get the following.

Lemma 6.22. *The system of equations* (6.5) *has a unique solution.*

In order to actually compute this, we employ a technique similar to the one used in Section 3.7.2 and derive a linear programming problem from the above system of equations (6.5). The only non-trivial step in this translation is to express the minimum operator. For this, we introduce additional variables $y_{q,a}$, which intuitively stand for $\min_{p\in\delta(q,a)} x_p$. Then (6.5) is transformed into

$$x_q = \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a\in\Sigma} y_{q,a} \quad (q \in Q). \tag{6.6}$$

To express the intuitive meaning of the variables $y_{q,a}$, we add the inequalities

$$y_{q,a} \leq x_p \quad \text{for all } q \in Q \text{ and } p \in \delta(q,a) \tag{6.7}$$

as well as the objective to maximize the values of these variables:

$$z = \max \sum_{q\in Q}\sum_{a\in\Sigma} y_{q,a}. \tag{6.8}$$

Lemma 6.23. *The LP problem consisting of the equations* (6.6)*, the inequations* (6.7)*, and the objective* (6.8) *has the unique solution*

$$\{x_q \mapsto \mu(q) \mid q \in Q\} \cup \{y_{q,a} \mapsto \min_{p\in\delta(q,a)} \mu(p) \mid p \in Q, a \in \Sigma\}.$$

Proof. Initially, observe that the above vector is in the feasible region, since it satisfies the restrictions (6.6) and (6.7). Next, we procede to show that it is indeed the only point that maximizes the objective function. First, we need the following claim.

Claim 6.24. *If* $\mathbf{x}$ *is a solution that maximizes the objective function then, for every* $q \in Q$ *and every* $a \in \Sigma$*, at least one of the inequalities* (6.7) *holds as an equality.*

Proof of Claim. Suppose on the contrary that $\mathbf{x}$ is a solution that maximizes z, but for some $q \in Q$ and $a \in \Sigma$, inequalities $x_{q,a} \leqslant \mu_p$ are strict for all $p \in \delta(q,a)$. This would mean that the value of $x_{q,a}$ can be increased, until it actually becomes equal to $\min_{p\in\delta(q,a)} \mu_p$, and all inequalities would still hold. The only restriction that would be hurt, is the one of the form (6.6) for the state q. This can be easily mended by setting μ_q to be equal to the right-hand side. This change will not affect any of the other restrictions. Thus, a new point $\mathbf{x}'$ has been produced, that satisfies all the restrictions of the LPP and additionally gives a larger value for the objective function. This is a contradiction to our initial assertion about $\mathbf{x}$. This completes the proof of the claim.

As a result, any points that are solutions to the LP problem, satisfy the condition

$$y_{q,a} = \min_{p\in\delta(q,a)} x_p \text{ for all } q, a.$$

Given that they also satisfy the equality constraints (6.6) (since they are in the feasible region), they correspond to solutions of the system of equations (6.5).

Finally, since there is a unique such solution, the solution of the LP problem is this unique solution. □

Since LP problems can be solved in polynomial time and the size of the LP problem in the above lemma is polynomial in the size of $\mathcal{A}$, we obtain an EXPTIME-upper bound for the computation problem and the decision problem. EXPTIME-hardness can again be shown by a reduction of the strict problem.

Even though the main idea is the same, the formal proof is quite more technical in this case. Initially note that solving (6.5) induces a "best" q-run of the automaton for every q; in every step, pick the state with the minimum value among all possible. Given such a best run of the automaton, we say that p is a *descendant* of q at depth d, if there are states $q_0 := q, q_1, \ldots, q_{d-1}, q_d := p$, and a word $a_1 \ldots, a_d$ s.t. $q_i = \arg\min_{q \in \delta(q_{i-1}, a_i)} \mu(q)$ for $i = 1, \ldots, d$. A *bad descendant* of a state q is a state $p \in F_a$ that is a descendant of q. Note that, if $q \in F_a$, then $\mu(q) \geqslant \frac{1}{2}$ and if $q \notin F_a$, then $\mu(q) \leqslant \frac{1}{2}$.

Lemma 6.25. *For a state q it holds that $\mu(q) > 0$ if and only if q has a bad descendant.*

Proof. If q has a bad descendant p, say at depth d, then there is a branch with nodes labeled $q, q_1, \ldots, p$ and thus $\mu(q) \geqslant \frac{1}{2|\Sigma|}\mu(q_1) \geqslant \cdots \geqslant (\frac{1}{2|\Sigma|})^d \mu(p) \geqslant (\frac{1}{2|\Sigma|})^d \frac{1}{2} > 0$.

Conversely, suppose that q has no bad descendant. Thus $q \notin F_a$ and the same holds for all its descendants. Then it holds that

$$\begin{aligned}\mu(q) &= \frac{1}{2}\chi_{F_a}(q) + \frac{1}{2|\Sigma|}\sum_{a \in \Sigma} \min_{p \in \delta(q,a)} \mu(p) \\ &\leqslant \frac{1}{2|\Sigma|}\sum_{a \in \Sigma} \max_{a \in \Sigma} \min_{p \in \delta(q,a)} \mu(p) \\ &= \frac{1}{2}\mu(p)\end{aligned}$$

for some child p of q. Iterating this for d steps, we get that $\mu(q) \leqslant (\frac{1}{2})^d \mu(p')$ for some descendant p' of q. But since $p' \notin F_a$, $\mu(p') \leqslant \frac{1}{2}$ and thus $\mu(q) \leqslant (\frac{1}{2})^{d+1}$. Since this holds for every d, it can be concluded that $\mu(q) = 0$. □

Lemma 6.26. *If q has a bad descendant, then q has a bad descendant at depth at most $|Q|$.*

Proof. Set $k = |Q|$. Suppose that there exists a $q_0 \in Q$ with no bad descendants up to depth k. It will be proved that there is a q_0-run with no states from F_a, i.e. q_0 has no bad descendants. No bad descendants up to depth k implies that there is a partial run of length k of $\mathcal{A}$ starting with q_0 that does not have state of F_a. For a branch of length k, the nodes are labelled with states $q_0, q_1, \ldots, q_k$. Since there are only k states, there are indices $i < j \leqslant k$ such that $q_i = q_j$. The tree having as root the node labeled with q_i has bigger length than the one labeled with q_j. Replace the latter tree with the former one. Then, all branches passing through the node labeled with q_j have length at least $k + 1$. Iterating this procedure for all branches, a partial run of length at least $k + 1$ is derived. Every time the above is repeated, a longer partial run is derived. We conclude that a partial run of infinite length, i.e. a q_0-run of $\mathcal{A}$ can be derived that has no states from F_a. Thus q_0 has no bad descendants. □

Lemma 6.27. *For the distance d_2, the decision problem for approximately solving language equations is* ExpTime*-hard.*

Proof. If $\mu(q) > 0$, then q has a bad descendant at depth at most $|Q|$. Thus it holds that $\mu(q) \geqslant (\frac{1}{2|\Sigma|})^{|Q|} \cdot \frac{1}{2} =: t$. We conclude that the decision problem with threshold t has a positive answer for the equation $\phi \approx^? \emptyset$ iff the equation $\phi =^? \emptyset$ has a solution. Since the latter problem is ExpTime-complete, we get an ExpTime-hardness result for our problem as well. □

Theorem 6.28. *For the distance d_2 and a polytime computable combining function, the computation problem (for approximate $\mathcal{FL}_0$ unification and for approximately solving language equations) can be solved in exponential time, and the decision problem is* ExpTime*-complete.*

For this theorem to hold, the exact definition of the distance d_2 is actually not important. Our approach works as long as the distance induces a system of equations similar to (6.5) such that Banach's fixed point theorem ensures the existence of a unique solution, which can be found using linear programming.

As an example, we can consider a weighted version of d_2, where different letters have different weights (degrees of importance). Given a weight function $wt\colon \Sigma \to [0,1]$, such that $\sum_{a\in\Sigma} wt(a) = 1$, extend this to a function over Σ^*, by setting $wt(u) = \prod_{i=1}^{k} wt(a_i)$ for $u = a_1 \dots a_k \in \Sigma^*$. Then, for $\nu, \lambda \in [0,1]$, define

$$\mu'(L) = \nu \sum_{u\in L} \lambda^{|u|} wt(u).$$

The condition that a difference for the word u counts as much as the sum of all differences for words uv properly extending u holds if we set $\lambda = \frac{1}{2}$. Furthermore, note that for $\nu = \lambda = \frac{1}{2}$, $wt(a) = \frac{1}{|\Sigma|}$ for every $a \in \Sigma$, we get the function μ defined for d_2.

6.4 On computing unifiers and a variation of the decision problem

The above procedures are able to check the *existence* of approximate solutions by computing the optimal degree of violation, but do not actually provide such *instances* of such optimal assignments.

For the distances d_1 and d_2 this seems to be possible. The main idea is that while solving the relevant optimization problems on tree automata, one could actually mark the transitions that lead to an optimal assignment. Hence, by only keeping these transitions it seems feasible to construct an LTA $\mathcal{A}_{opt}$, every run of which will correspond to an optimal assignment. However, the exact details of this construction have to be properly investigated.

Furthermore, in order to obtain approximate unifiers, we need to be able to obtain optimal *finite* assignments. By applying the results of Lemma 3.43 on the above automaton $\mathcal{A}_{opt}$, we can obtain a BTA with a successful run for every such assignment. However, it is not clear how to obtain a near-optimal solution in case that no finite assignments achieves the optimal value.

Finally, if we are able to answer the above questions, we will also obtain an algorithm for the variation of the decision problem where $<$ is replaced by $\leq$, that is, given a threshold p

we want to check whether there exists a substitution σ such that $\nu_m(\sigma, C, D) \leq p$. ExpTime-hardness of this problem is clear (even for a unary encoding of the threshold p), since we can test exact unification by setting $p = 0$.

This problem can no longer be reduced to the computational problem itself. However, if we have some extra knowledge on whether the infimum is indeed a minimum, that is, if there exists a substitution that achieves the optimal degree of violation, then we can once again employ the computational problem to solve this variation of the decision problem. In particular, we have the following result in the lines of Lemma 6.2.

Lemma 6.29. *Let m be a concept distance, C, D $\mathcal{FL}_0$ concept patterns, and $p \in \mathbb{Q}$ a threshold. Then there exists a substitution $\sigma \in Sub$ with $\nu_m(\sigma, C, D) \leq p$ iff*

- $\inf_{\sigma \in Sub} \nu_m(\sigma, C, D) \leq p$ *and this infimum is actually a minimum, or*
- $\inf_{\sigma \in Sub} \nu_m(\sigma, C, D) < p$ *and this infimum is not a minimum.*

Hence, in order to be able to answer this decision problem as well, it suffices to be able to determine whether the value obtained from the computational problem is achievable by a substitution or not.

For the corresponding decision problem for approximate language equations, the complete analogous result holds.

Investigating in detail the problems mentioned in this section is a natural extension of the results obtained in this chapter and is appropriate for future work.

Chapter 7

Approximate matching

Matching is the special case of unification where one of the terms to be unified has no variables and thus remains unchanged under substitutions. In Description Logics (DLs), matching concepts against patterns was introduced to help filter out unimportant aspects of complicated concepts appearing in large industrial knowledge bases [BM96]. For $\mathcal{FL}_0$, matching was investigated in detail in [BN01]. As it is a special case of unification, the results of Section 4.3 apply in this setting, as well. In particular, checking whether a matcher between a concept description and a pattern exists reduces to checking solvability of language equations. However, since no variables occur in one of the sides, we don't have to deal with equations of the general form (4.4), but of a rather simpler kind. This allows to solve the problem in polynomial time, in contrast to unification, which is ExpTime-complete, as we have seen in Section 4.3.

In this section, we will prove that a similar result holds for the approximate setting: approximate matching is of lower complexity than approximate unification. To this purpose, we need to solve language equations of the form (7.1) (introduced below) approximately. Initially, we describe the algorithm used in [BN01] to check solvability of matching (in the classical setting) in polynomial time. Next, after we formally define approximate matching, we will prove that the problem is in NP for a wide range of distance measures. Finally, we will investigate the distances d_0, d_1 and d_2 in more detail, proving that the problem for d_1 is actually solvable in polynomial time, while it is NP-complete for the other two. For the latter, we will make use of Max-±Pos-SAT, an NP-complete satisfiability problem. To the best of our knowledge, this problem has not been investigated in the literature before, hence we devote a section at the end of this chapter to formally introduce it and prove its NP-completeness.

7.1 Classic and approximate matching

The definition of matching is the same as Definition 4.7 for unification, with the difference that one of the patterns contains no variables, in other words it is a concept description. More precisely, we have the following.

Definition 7.1 (Matching). *Let C be an $\mathcal{FL}_0$ concept description, D an $\mathcal{FL}_0$ concept pattern, and $\mathcal{T}$ an $\mathcal{FL}_0$ TBox. The substitution σ is a* matcher *of C, D w.r.t. $\mathcal{T}$ if $C \equiv_{\mathcal{T}} \sigma(D)$. An $\mathcal{FL}_0$* matching problem *$C \equiv^?_{\mathcal{T}} D$ asks whether given such C, D and $\mathcal{T}$ there exists a matcher w.r.t. $\mathcal{T}$ or not.* ♢

As was the case for unification, previous results only deal with the case of the empty TBox. In the rest of this chapter, we will impose the same restriction, and defer the study

of TBoxes for the next chapter. Now, after we review the approach from [BN01], we will extend matching (of this restricted case) to the approximate setting.

Since matching a special case of unification, all the results obtained in Section 4.3 apply to this simpler setting as well. In particular, analogously to (4.4), matching can be reduced to formal language equations of the following form: given finite languages $K_0, L_0, L_1, \ldots, L_m$ we want to know whether there exist finite languages $X_1, \ldots, X_m$ such that

$$K_0 = L_0 \cup L_1 \cdot X_1 \cup \cdots \cup L_m \cdot X_m. \tag{7.1}$$

As before, a *solution* of such an equation is an *assignment* σ of languages to the variables X_i such that the above equation holds as equality of languages. An assignment is called *finite* if all the languages $\sigma(X_i)$ are finite. Identically to (4.5), the assignments are required to be finite in order to maintain the correspondence with the Description Logic setting. Hence, by slight abuse of notation, we will use the term *matcher* to also denote a finite assignment that is a solution. Overall, matching in $\mathcal{FL}_0$ reduces to checking whether equations of the form (7.1) have a matcher. Baader and Narendran [BN01] showed that this problem is decidable in polynomial time by proving that (7.1) has a matcher iff the assignment

$$\theta(X_i) := \bigcap_{v \in L_i} v^{-1} K_0 \quad (i = 1, \ldots, m)$$

is a solution of (7.1). It is easy to see that computing θ and checking whether it actually is a solution can be done in polynomial time.

As an example, consider the equation

$$\{a, ab, abb\} = \{a, ab\} \cdot X.$$

Following the procedure above, for $m = 1$, $K_0 = \{a, ab, abb\}$, $L_0 = \emptyset$ and $L_1 = \{a, ab\}$ we have that $a^{-1}K_0 = \{\varepsilon, b, bb\}$, $(ab)^{-1}K_0 = \{\varepsilon, b\}$ and thus $\theta(X) := \{\varepsilon, b, bb\} \cap \{\varepsilon, b\} = \{\varepsilon, b\}$. Immediately, it can be verified that $\{a, ab\} \cdot \theta(X) = \{a, ab, abb\}$ holds as equality, and thus θ is a solution.

On the other hand, for the equation

$$\{ab\} = \{a, ab\} \cdot X$$

working as before we obtain $a^{-1}K_0 = \{b\}$, $(ab)^{-1}K_0 = \{\varepsilon\}$ and thus $\theta(X) := \{b\} \cap \{\varepsilon\} = \emptyset$. Since $\{a, ab\} \cdot \theta(X) \neq \{ab\}$, this problem does not have a solution.

As we have shown in the previous chapter, approximate unification in $\mathcal{FL}_0$ w.r.t. d_0, d_1 and d_2 (or rather, w.r.t. the concept distances induced by these language distances and a proper combining function) is of the same complexity as exact unification, i.e., ExpTime-complete. We will now see that for approximate matching, the exact complexity depends on the distance used, but, in any case, remains lower than the corresponding bound for unification. First, let us formally define the problem. Again, our techniques only apply for the case of the empty TBox. Therefore, right from the start we define the problem in this restricted setting.

Definition 7.2 (Approximate matching). *Given concept description C, concept pattern D, CDM m and threshold $p \in \mathbb{Q}$ the approximate matching problem asks whether there exists a substitution σ such that*

$$m(C, \sigma(D)) < p.$$

If such a substitution exists, it is called a p-approximate matcher *of* C, D. ♢

As for approximate unification, this *decision problem* reduces (in polynomial time) to the respective *computational problem*, that is, to compute $\inf_{\sigma \in Sub} m(C, \sigma(D))$.

We will not deviate from the pattern established in the previous chapter; we will reduce the above problem to one involving formal languages.

Definition 7.3 (Approximate language matching). *Given an equation of the form* (7.1)*, the* approximate language matching problem *w.r.t. the language distance d with threshold $p \in \mathbb{Q}$ asks whether there exists a finite assignment σ such that*

$$d(K_0, L_0 \cup L_1\sigma(X_1) \cup \cdots \cup L_m\sigma(X_m)) < p.$$

Such an assignment, if one exists, is called a p-approximate matcher. *A (not necessarily finite) assignment that satisfies the above inequality is called a* p-approximate solution. ♢

The same relation between this decision problem and the corresponding computational problem holds in this setting, as well.[1] As we will see in the next section, however, we can also use the decision problem to answer the computation problem for a wide class of language distances, that includes d_0, d_1 and d_2. Furthermore, by Lemma 6.5 we obtain that the computational problem for approximate $\mathcal{FL}_0$ matching reduces to the computational (and hence also the decision) problem for approximate language matching.

Returning to our previous example about the equation $\{ab\} = \{a, ab\} \cdot X$, w.r.t. d_1 with threshold 2^{-2} we have that the assignment $\sigma_1(X) = \{b\}$ is a 2^{-2}-approximate matcher while $\sigma_2(X) = \emptyset$ is not, since

$$d_1(\{ab\}, \{a, ab\} \cdot \sigma_1(X)) = d_1(\{ab\}, \{ab, abb\}) = 2^{-3} < 2^{-2} \text{ while}$$
$$d_1(\{ab\}, \{a, ab\} \cdot \sigma_2(X)) = d_1(\{ab\}, \emptyset) = 2^{-2} \not< 2^{-2}.$$

Furthermore, any assignment σ with $\{b\} \subseteq \sigma(X) \subseteq \{a, b\}^* \setminus \{\varepsilon, a\}$ is an approximate solution for the same threshold.

7.2 Containment in NP

In this section, we prove that, for language distances that are monotone w.r.t. the symmetric difference of the input languages, i.e., that satisfy property (3.1) introduced in Section 3.3, the existence of a p-approximate solution implies the existence of a p-approximate matcher of polynomial size. Recall that the property requires $K \Delta L \subseteq M \Delta N \implies d(K, L) \leq d(M, N)$ for any languages K, L, M, N, and it holds for all language distance functions of the form $d(K, L) = f(K \Delta L)$ where f is a monotone function. If the distance is computable in polynomial time, this yields an NP-algorithm for deciding the approximate matching problem.

[1] Recall the discussion in Section 6.1.

Assume that we are given an equation of the form (7.1), a distance d satisfying property (3.1), and a threshold p, and let σ be a p-approximate solution. The following observation suffices to prove our claim:

Observation. *Let $F_i := \bigcup_{w \in L_i} w^{-1}K_0$. Then, $\sigma'(X_i) = \sigma(X_i) \cap F_i$ is also a p-approximate matcher.*

Indeed, assume that $u \in \sigma(X_i) \setminus F_i$. Then $wu \notin K_0$ for every $w \in L_i$. Thus we get

$$K_0 \Delta \sigma' \left(L_0 \cup \bigcup_{i=1}^{m} L_i X_i \right) \subseteq K_0 \Delta \sigma \left(L_0 \cup \bigcup_{i=1}^{m} L_i X_i \right)$$

and hence

$$d\left(K_0, \sigma' \left(L_0 \cup \bigcup_{i=1}^{m} L_i X_i \right) \right) \leq d\left(K_0, \sigma \left(L_0 \cup \bigcup_{i=1}^{m} L_i X_i \right) \right) < p,$$

i.e., σ' is a p-approximate matcher as well (in fact a better one). Note that for every i we have that $\sigma'(X_i) \subseteq F_i \subseteq Suf(K_0)$, where $Suf(w) = \{u \in \Sigma^* \mid \exists v \in \Sigma^*.vu = w\}$ and $Suf(L) = \bigcup_{w \in L} Suf(w)$. Note that $|Suf(L)| \leq ||L|| + |L|$, which is linearly bounded by the size of L. Furthermore, the longest word in $Suf(L)$ is bounded by the size of $||L||$. Hence, we can conclude that $Suf(K_0)$, and thus also $\sigma'(X_i)$ for every i, is at most quadratic in the size of K_0, and overall we obtain the following.

Lemma 7.4. *Assume that d is a language distance function satisfying property* (3.1)*. An approximate matching problem w.r.t. d with threshold p has a p-approximate solution iff it has a p-approximate matcher of size at most quadratic in the size of the problem.*

If the distance is computable in polynomial time, by guessing an assignment of polynomial size and then checking in polynomial time whether it actually is a p-approximate matcher for the given threshold value, we obtain an NP-algorithm for approximate matching. Overall, we have the following result.

Theorem 7.5. *Given an equation of the form* (7.1)*, a threshold p, a distance d satisfying property* (3.1) *that is computable in polynomial time, the approximate language matching problem is decidable in NP.*

A further consequence of Lemma 7.4 is that we can use the decision problem to also answer the computational problem of approximate language matching. More specifically, for language distances as in Theorem 7.5, we have that the (representation of the) value of d is polynomial in the size of the input languages, since they are computable in polynomial time. Furthermore, by (the proof of) Lemma 7.4 we obtain that the search space for a potential matcher is not only finite, but also consists of substitutions of polynomial size. Hence, the infimum among all degrees of violation is actually a minimum, and this minimum is of size polynomial in the input problem. Overall, by applying binary search we can call the algorithm for the decision problem a polynomial number of times to compute the optimal value achievable.

7.3 Approximate Matching w.r.t. d_1

For the language distance d_1, we can actually get a better complexity result: the approximate matching problem is decidable in polynomial time.

Since d_1 is monotone w.r.t. the symmetric difference of the input languages, it suffices to check whether there is an approximate solution (that is not necessarily finite). In the positive case, Lemma 7.4 then guarantees the existence of an approximate matcher of polynomial size, and hence finite. Looking at the definition of d_1, it is easy to see that, if the input languages agree on all words of length up to $m-1$, their distance is at most 2^{-m}. More generally, we have the following:

Lemma 7.6. *Let K, L be languages over Σ and $p \leq 2^{-m}$, $m \in \mathbb{N}$. Then,*

$$d_1(K,L) < p \iff K \cap \Sigma^{\leq m} = L \cap \Sigma^{\leq m} \iff K \cup \Sigma^{\geq m+1} = L \cup \Sigma^{\geq m+1}.$$

As an easy consequence of this lemma, we obtain:

Proposition 7.7. *Given an equation of the form* (7.1)*, the assignment $\sigma(X_i) = M_i$ is a p-approximate solution w.r.t. d_1 with threshold $p \leq 2^{-m+1}$ iff*

$$K_0 \cup \Sigma^{\geq m} = L_0 \cup L_1 M_1 \cup \cdots \cup L_m M_m \cup \Sigma^{\geq m},$$

i.e., iff it is a solution of the equation

$$K_0 \cup \Sigma^{\geq m} = L_0 \cup L_1 X_1 \cup \cdots \cup L_m X_m \cup \Sigma^{\geq m}. \tag{7.2}$$

♢

Reflecting on the equation $\{ab\} = \{a, ab\}X$ w.r.t. d_1 with threshold $p = 2^{-2}$ we have that

$$\{ab\} \cup \Sigma^{\geq 3} = \{ab, abb\} \cup \Sigma^{\geq 3} = \{a, ab\}\,\sigma_1(X) \cup \Sigma^{\geq 3},$$

thus verifying again that σ_1 is a 2^{-2}-approximate solution. In fact, for any assignment σ with $\{b\} \subseteq \sigma(X) \subseteq \{a,b\}^* \setminus \{\varepsilon, a\}$ it holds that

$$\{ab\} \cup \Sigma^{\geq 3} = \{a, ab\}\,\sigma_1(X) \cup \Sigma^{\geq 3}.$$

Meanwhile,

$$\{ab\} \cup \Sigma^{\geq 3} \neq \emptyset \cup \Sigma^{\geq 3} = \{a, ab\}\,\sigma_2(X) \cup \Sigma^{\geq 3},$$

since σ_2 is not a 2^{-2}-approximate solution.

By Proposition 7.7, finding a p-approximate solution w.r.t. d_1 with $p \leq 2^{-m+1}$ for the equation of the form (7.1) reduces to finding a solution for the equation (7.2). Adapting the technique used in [BN01] for matching in $\mathcal{FL}_0$, we get the following result.

Lemma 7.8. *An equation of the form* (7.2) *has a solution iff*

$$\sigma(X_i) = M_i := \bigcap_{v \in L_i} \left(v^{-1}(K_0 \cup \Sigma^{\geq m})\right)$$

is a solution.

Proof. The if direction is trivial.

For the only-if direction, assume that $\tau(X_i) = M_i$ is a solution of (7.2). We want to show that

$$K_0 \cup \Sigma^{\geq m} = L_0 \cup L_1 M_1 \cup \cdots \cup L_m M_m \cup \Sigma^{\geq m}. \tag{7.3}$$

Obviously, the equation holds for all words of length at least m. Suppose that $w \in K_0$ and $|w| < m$. Since τ is a solution, either $w \in L_0$ or $w \in L_i M_i$ for some $i \in \{1, \ldots, m\}$. In the first case, there is nothing more to show. In the second case, there are words $v_0 \in L_i, u \in M_i$ s.t. $w = v_0 u$. Since τ is a solution, for every word $v \in L_i$ it holds that $vu \in K_0 \cup \Sigma^{\geq m}$. Thus $u \in \bigcap_{v \in L_i} v^{-1}(K_0 \cup \Sigma^{\geq m}) = M_i$, which proves language inclusion in one direction.

For the other direction, since there is a solution, it holds that $L_0 \subseteq K_0 \cup \Sigma^{\geq m}$. Thus it suffices to prove that $L_i M_i \subseteq K_0 \cup \Sigma^{\geq m}$. Assume that $w \in L_i M_i$. This means that $w = v_0 \ell$ for some $v_0 \in L_i$ and $\ell \in M_i$. Thus we get

$$\ell \in \bigcap_{v \in L_i} v^{-1}(K_0 \cup \Sigma^{\geq m}) \implies \ell \in {v_0}^{-1}(K_0 \cup \Sigma^{\geq m}) \implies v_0 \ell \in K_0 \cup \Sigma^{\geq m},$$

which completes the proof. □

Applying the above lemma to our running example, we obtain

$$\begin{aligned} \sigma(X) :&= a^{-1}(\{ab\} \cup \{a,b\}^{\geq 3}) \cap (ab)^{-1}(\{ab\} \cup \{a,b\}^{\geq 3}) \\ &= (\{b\} \cup \{a,b\}^{\geq 2}) \cap (\{\varepsilon\} \cup \{a,b\}^{\geq 1}) \\ &= (\{b\} \cup \{a,b\}^{\geq 2}) = \{a,b\}^* \setminus \{\varepsilon, a\}, \end{aligned}$$

which we have already seen that is a 2^{-2}-approximate solution.

Checking in polynomial time whether the assignment from Lemma 7.8 is actually a solution can be done by using tree-like automata (see [BN01] for details). Overall, we obtain the following for d_1.

Theorem 7.9. *Given an equation of the form* (7.1)*, the approximate matching problem w.r.t. d_1 with threshold p is decidable in polynomial time.*

7.4 Hardness for d_0 and d_2

In the following, we prove that approximate matching w.r.t. d_0 and d_2 is NP-hard for $\mathcal{FL}_0$. We provide a reduction of an (arbitrary) instance of Max-±Pos-2SAT (see Section 7.5 for details, as well as a proof of NP-hardness of that problem) to an instance of approximate matching w.r.t. m^{d_0} and m^{d_2}. Since the reduction will only require a single concept name, there is no need for a combining function f; equivalently, we can assume that identity is the combining function.[2] Hence we omit f from the notation of the distance.

[2] If a combining function f other than the identity is to be used, the same proof works by changing the threshold p we obtain below to $f(p)$.

Assume that we are given an instance I of Max-±Pos-2SAT, i.e., a formula $\Phi = \bigwedge_{i=1}^{n} f_i$ with f_i being of the form

$$\phi_1 = x_1 \vee x_2 \text{ and } \phi_2 = \neg(x_1 \vee x_2).^3 \tag{7.4}$$

over the variables $\mathcal{X} = \{x_1, \ldots, x_m\}$, and an integer p. Recall that the decision problem is whether exists a satisfying assignment ν, i.e., an assignment that satisfies at least p of the n conjuncts of Φ.

Intuitively, every conjunct will correspond to a chain of value restrictions, i.e., a word over $\mathsf{N_R}$. For positive formulas, this will be chains of value restrictions that appear in the concept description. If such a chain of value restrictions is not matched by the assignment on the concept pattern, meaning that the corresponding formula is violated, this word would contribute to the difference between the concepts. On the other hand, negative formulas will correspond to chains of value restrictions that do not appear in the concept description but might occur in the (description obtained after we apply a substitution to the) concept pattern, depending on the substitution. This will be the case if the corresponding formula is violated, and again the distance between the concepts will increase.

We construct the approximate matching problem I' w.r.t. m^{d_0} and m^{d_2} by setting

- $C = \forall K_0.A$, where $K_0 = \{a^{n-i}b^i \mid 1 \leq i \leq n \wedge f_i \text{ positive}\}$, and
- $D = \bigsqcap_{j=1}^{m} \forall L_j.X_j$, where $L_j = \{a^{n-i}b^i \mid x_j \in Var(f_i)\}$ for $j = 1, \ldots, m$,

thus obtaining the problem $C \equiv^? D$. For d_0 we set the threshold to $q = n - p + 1$, while for d_2 to $q = \frac{n-p+1}{2 \cdot 4^n}$.[4]

Lemma 7.10. *The original instance I has a satisfying assignment iff I' has a q-approximate matcher.*

Proof. For the only if direction, assume that I has a satisfying assignment $\nu : \mathcal{X} \to \{0,1\}$. Define the substitution $\sigma(X_i) = \forall M_i.A$, where

$$M_i = \begin{cases} \{\varepsilon\} & \text{if } \nu(x_i) = 1 \\ \emptyset & \text{otherwise} \end{cases}.$$

Note that

$$m^{d_0}(C, \sigma(D)) = d_0(K_0, \bigcup_{i=1}^{m} L_i M_i) = |K_0 \,\Delta \bigcup_{i=1}^{m} L_i M_i|, \text{ and}$$

$$m^{d_2}(C, \sigma(D)) = d_2(K_0, \bigcup_{i=1}^{m} L_i M_i) = \mu(K_0 \,\Delta \bigcup_{i=1}^{m} L_i M_i).$$

Obviously, $K_0 \,\Delta \bigcup_{i=1}^{m} L_i M_i \subseteq \{a^{n-i}b^i \mid 1 \leq i \leq n\}$. Making use of the following claim, the result is immediate.

Claim. $\nu(f_i) = 0 \iff a^{n-i}b^i \in K_0 \,\Delta \bigcup_{j=1}^{m} L_j M_j$.

[3] Formulas of the form ϕ_1 will be referred to as *positive*, while formulas of the form ϕ_2 will be referred to as *negative*.

[4] Note that, if encoded in binary, this threshold is linear in the size of the original instance.

Assume that f_i is positive, i.e., of the form $x_{i_1} \vee x_{i_2}$. Then, by construction of I', $a^{n-i}b^i \in K_0$. Observe that, on the right hand side, $a^{n-i}b^i$ is obtainable only if for at least one $j \in \{i_1, i_2\}$, X_j is nonempty.

Since $\nu(f_i) = 0$, this means that $\nu(x_j) = 0$ for $j = i_1, i_2$, which in turn implies that $M_j = \emptyset$ for $j = i_1, i_2$. Thus, $a^{n-i}b^i$ cannot be obtained on the right hand side, and thus is in the symmetric difference.

Likewise, if f_i is negative, $a^{n-i}b^i \notin K_0$. But $\nu(f_i) = 0$ now means that (at least) one of the x_j is evaluated to 1, thus implying that one of the M_j contains ε. This time, we obtain that $a^{n-i}b^i$ is on the right hand side, and thus is in the symmetric difference.

The reverse implication is completely analogous and the proof of the claim is complete.

Since ν is a satisfying assignment, there are at most $n-p$ formulas f_i such that $\nu(f_i) = 0$. From the claim, we obtain that $|K_0 \,\Delta \bigcup_{i=1}^{m} L_i M_i| \leq n - p < n - p + 1$, which is exactly the threshold for d_0. For d_2, since all words in the symmetric difference are of length n, $d_2(K_0, \bigcup_{j=1}^{m} L_j M_j) \leq \frac{n-p}{2 \cdot 4^n} < \frac{n-p+1}{2 \cdot 4^n}$, thus σ_ν is a q-approximate matcher.

For the if direction, assume that I' has a q-approximate matcher σ. From Lemma 7.4 and the observation that preceeds it, there is also a q-approximate matcher σ' such that $\sigma'(X_i) = \forall M_i.A$ with $M_i \subseteq \bigcup_{v \in L_i} v^{-1}K_0 = \{\varepsilon\}$ for every i.

Define the truth assignment $\nu_{\sigma'}(x_i) = 1 \iff \sigma'(X_i) = \{\varepsilon\}$. An argument completely analogous to the one used for the only if direction completes the proof. □

Example 7.11. *The above reduction can easily be understood with an example. Assume we are given the formula* $\Phi = \big(x_1 \vee x_3\big) \wedge \big(x_2 \vee x_3\big) \wedge \big(\neg(x_1 \vee x_3)\big) \wedge \big(\neg(x_1 \vee x_2)\big)$. *The chains of value restrictions corresponding to these conjuncts are respectively* a^3b, a^2b^2, ab^3, b^4. *We obtain*

- $K_0 = \{a^3b, a^2b^2\}$, *since the first and second formulas are positive,*
- $L_1 = \{a^3b, ab^3, b^4\}$, *since* x_1 *occurs in the first, third, and fourth formulas,*
- $L_2 = \{a^2b^2, b^4\}$, *since* x_2 *occurs in the second and fourth formulas, and*
- $L_3 = \{a^3b, a^2b^2, ab^3\}$, *since* x_3 *occurs in the first, second, and third formulas.*

One can easily verify that $F_1 = F_2 = F_3 = \{\varepsilon\}$. *An optimal assignment for the original problem is* ν *that sets* x_1, x_2, x_3 *to* $0, 0, 1$ *respectively, which only violates the third formula. This corresponds to the (optimal as well) substitution* σ_ν *with* $\sigma_\nu(X_1) = \sigma_\nu(X_2) = \top$, $\sigma_\nu(X_3) = A$, *which outputs* $\forall\{a^3b, a^2b^2, ab^3\}.A$ *in the right-hand side, with* $\forall ab^3.A$ *being the only difference with the left-hand side.* ♢

Thus, we have the main result of this section.

Theorem 7.12. *The* $\mathcal{FL}_0$ *approximate matching problem w.r.t.* d_0 *and* d_2 *is NP-complete.*

7.5 Max-±Pos-(n)SAT

In this section, we formally define the problem Max-±Pos-(n)SAT, which we already used in the previous section, and we prove that it is NP-complete via a reduction from the Max-Cut problem, which is known to be NP-complete [GJ90].

We consider clauses of positive literals and negations of such clauses, i.e., formulas of one of the following forms:

$$\phi_1 = \bigvee_{i=1}^{k} x_i \text{ and } \phi_2 = \neg\left(\bigvee_{i=1}^{\ell} x_i\right). \tag{7.5}$$

Formulas of the form ϕ_1 will be referred to as *positive*, while formulas of the form ϕ_2 will be referred to as *negative*.

Definition 7.13. *Let $v : \mathcal{X} \to \{0,1\}$ be a truth assignment of the propositional variables $\mathcal{X} = \{x_1, \ldots, x_m\}$. Given a formula f over $\mathcal{X}$, we define $Var(f)$ to be the set of variables occurring in f. If f is of the form ϕ_1, we have that $v(f) = 1$ iff for at least one $x \in Var(f)$ it holds that $v(x) = 1$. If f is of the form ϕ_2, we have that $v(f) = 1$ iff for every $x \in Var(f)$ it holds that $v(x) = 0$.*

Given a formula $\Phi = \bigwedge_{i=1}^{n} f_i$ with conjuncts of the form (7.5)*, the* satisfiability problem for positive clauses and their negations (±Pos-SAT) *asks whether there is a satisfying assignment, i.e., an assignment of truth values to the variables s.t. $v(f_i) = 1$ for all $i = 1, \ldots, m$. The special case when the formulas f_i are restricted to contain n variables is called* ±Pos-nSAT. ♢

It is obvious that ±Pos-SAT is solvable in linear time. Indeed, one just has to set all variables that appear in a negative formula to 0, and then check whether any of the formulas of the form ϕ_1 evaluates to 0 under this assignment. However, we prove that the optimization version of ±Pos-SAT, that we call Max-±Pos-SAT, is NP-hard. In fact, we will prove it already for the special case *Max-±Pos-2SAT*, where the formulas f_i are restricted to contain 2 variables.

Definition 7.14. *The problem Max-±Pos-2SAT is defined as follows: given a formula $\Phi = \bigwedge_{i=1}^{n} f_i$ with conjuncts f_i of the form* (7.5) *with $k = \ell = 2$ over the variables $\mathcal{X}$, and an integer N, determine whether there exists an assignment $v : \mathcal{X} \to \{0,1\}$ that satisfies at least N conjuncts of Φ.* ♢

Proposition 7.15. *The problem Max-±Pos-2SAT is NP-complete.*

Proof. Containment in NP is obvious by a guess-and-check algorithm. We prove hardness using a reduction from the Max-Cut problem. Since Max-Cut is an NP-complete problem [GJ90], we obtain the overall result.

Let $G = (V, E)$ be an undirected graph, for which we want to know whether there exists a partition $V_0 \uplus V_1 = V$ of V such that there are at least $N > 0$ edges $\{u, v\}$ with $u \in V_0$ and $v \in V_1$.

We build a corresponding instance Φ of Max-±Pos-2SAT as follows. The variables of Φ are $V \uplus E$. Let $\psi(x, y, z)$ be the following ±Pos-2SAT formula: $(x \vee y) \wedge (x \vee y) \wedge \neg(x \vee z) \wedge \neg(y \vee z)$. For every edge $e = \{u, v\}$ in G, we add the conjuncts of $\psi(u, v, e)$ to Φ. We show that G has a cut with at least N crossing edges if, and only if, Φ has an assignment that satisfies at least $N + 2|E|$ conjuncts.

Let $V_0 \uplus V_1 = V$ be a cut of G with at least N crossing edges. Consider the following assignment of the variables of Φ to $\{0,1\}$. A variable corresponding to a vertex $v \in V$ is mapped to $b \in \{0,1\}$ if, and only if, $v \in V_b$. A variable corresponding to an edge $e = \{u, v\}$ is mapped to b if both u and v are in V_b, and is mapped to an arbitrary value if u and v are in different sets of the partition. For every edge $e = \{u, v\}$, the following cases occur:

- If $u, v \in V_0$, then the first two conjuncts of $\psi(u, v, e)$ are violated, while the last two conjuncts are satisfied.
- If $u, v \in V_1$, then the first two conjuncts of $\psi(u, v, e)$ are satisfied, and the last two conjuncts are violated.
- If u, v are in different parts, then the first two conjuncts of $\psi(u, v, e)$ are satisfied and exactly one of the other two conjuncts is satisfied.

Therefore, exactly $3N + 2(|E| - N) = N + 2|E|$ conjuncts of Ψ are satisfied.

Conversely, consider an assignment $s\colon V \uplus E \to \{0, 1\}$ of the variables of Φ that satisfies at least $N + 2|E|$ conjuncts of Φ. Define a partition $V_0 \uplus V_1 = V$ of V by taking $v \in V_b$ if, and only if, v is mapped to b by the assignment. We claim that this defines a cut with at least N crossing edges. Note that the maximum amount of satisfiable constraints of ψ is 3, which is realized by any assignment that assigns different values to x and y, and that if x and y are assigned the same value, then exactly 2 constraints are satisfied. Suppose that fewer than N edges cross. This means that fewer than N constraints of the form $\psi(u, v, e)$ are such that $s(u)$ and $s(v)$ are distinct. As we noted above, exactly 2 conjuncts of $\psi(u, v, e)$ are satisfied if $s(u)$ and $s(v)$ are equal. This means that only $2|E|$ conjuncts of Φ are satisfied by s, a contradiction to the hypothesis that s satisfies at least $N + 2|E|$ conjuncts. □

7.6 Outlook

We have shown in this chapter that the problem of approximate matching in $\mathcal{FL}_0$ w.r.t. the empty TBox is contained in NP for a wide class of distances. Furthermore, we obtained sharp bounds for certain language distances, proving the problem to be solvable in polynomial time for d_1, while it is NP-complete for d_0 and d_2. In order to achieve the latter hardness results we devised an NP-complete satisfiability problem, which, to the best of our knowledge is novel.

Matching in extensions of $\mathcal{FL}_0$ has been investigated in [BKB+99], where the problem reduces to solving equations that involve infinite languages. Hence, we would like to investigate whether the results of this chapter can be extended to such equations.

In the next chapter, we will return to the classic (non-approximate) setting, in order to incorporate TBoxes into matching. It would be interesting to explore whether this is possible for the approximate case as well.

Chapter 8

Matching in $\mathcal{FL}_0$ w.r.t. TBoxes

Until now, we extended unification to the approximate setting in Chapter 6. Furthermore, we showed in Chapter 7 that the result of matching being of lower complexity than unification transfers to this setting as well. Following the program we set out in the beginning of this thesis, it is now time to extend to the other direction, that is, incorporate TBoxes in this reasoning service. However, we have no guarantee that unification in the presence of TBoxes is actually decidable. In fact, previous results do not provide any hint on anything else than unfoldable TBoxes. It hence makes sense to investigate the special case of matching. Since it is of lower complexity in both the classic and the approximate case, it is intuitively more probable that we can achieve a positive result. This intuition turns out to be correct, as we demonstrate below.

In this chapter, we show that matching in $\mathcal{FL}_0$ in the presence of general TBoxes is an ExpTime-complete problem. Since already subsumption in $\mathcal{FL}_0$ w.r.t. TBoxes is ExpTime-complete [BBL05], ExpTime-hardness of this problem is clear. The first main contribution of this chapter is thus to show the ExpTime upper bound. We do this by first showing an ExpTime upper bound for the problem of testing whether an $\mathcal{FL}_0$ matching problem has a matcher in the extended logic $\mathcal{FL}_{\text{reg}}$. Basically, in $\mathcal{FL}_{\text{reg}}$ one can use regular languages to express infinite conjunctions of value restrictions. Our proof of the ExpTime upper bound depends on a fine-grained analysis of the complexity of subsumption of $\mathcal{FL}_{\text{reg}}$ concept descriptions w.r.t. an $\mathcal{FL}_0$ TBox. The second step is then to show that an $\mathcal{FL}_0$ matching problem has an $\mathcal{FL}_0$ matcher iff it has an $\mathcal{FL}_{\text{reg}}$ matcher. The second main contribution of this chapter is to show that the complexity of the matching problem can be lowered from ExpTime to PSpace if one considers TBoxes of a restricted form where the role depth on the left-hand side of a GCI is not larger than the role depth on the right-hand side.

8.1 The description logic $\mathcal{FL}_{\text{reg}}$

The DL $\mathcal{FL}_{\text{reg}}$ extends $\mathcal{FL}_0$ by allowing the use of regular languages L over the alphabet of all role names $\mathsf{N_R}$ to express infinite conjunctions of value restrictions. Basically, the value restriction $\forall L.C$ stands for the (possibly infinite) conjunction $\bigsqcap_{w\in L} \forall w.C$, where (for $w = r_1 \dots r_k \in \mathsf{N_R}^*$) the expression $\forall w.C$ is an abbreviation for $\forall r_1.\cdots\forall r_k.C$. To be more precise, the set $\mathcal{C}_{\mathcal{FL}_{\text{reg}}}(\mathsf{N_C}, \mathsf{N_R})$ of *$\mathcal{FL}_{reg}$ concept descriptions over the set of concept names* $\mathsf{N_C}$ *and the set of role names* $\mathsf{N_R}$ is obtained using the following syntax rules:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall L.C,$$

where $A \in \mathsf{N_C}$, L is a regular language over $\mathsf{N_R}$, and $C \in \mathcal{C}_{\mathcal{FL}_{\text{reg}}}(\mathsf{N_C}, \mathsf{N_R})$. Again, if there is no fear of confusion, we will usually omit the sets of concept and role names (also from the notation) and simply refer to the set of $\mathcal{FL}_{\text{reg}}$ concept descriptions ($\mathcal{C}_{\mathcal{FL}_{\text{reg}}}$). Since the singleton language $\{r\}$ for $r \in \mathsf{N_R}$ is regular, every $\mathcal{FL}_0$ concept description is also an $\mathcal{FL}_{\text{reg}}$ concept description.

The size of an $\mathcal{FL}_{\text{reg}}$ concept description is defined similarly to the case of $\mathcal{FL}_0$, but we need to further define the size of a concept description of the form $C = \forall L.D$. Here we assume that the regular language L is given by a regular expression or an NFA. If the size of the representation of L is n, then $|C| := n + |D|$.

Semantics is again defined using interpretations, as was done in Definition 4.1 for $\mathcal{FL}_0$. Once more, we need to define how the interpretation function is extended to concept descriptions of the form $\forall L.C$. This is done as follows:

- $(\forall \varepsilon.C)^{\mathcal{I}} := C^{\mathcal{I}}$,
- $(\forall w.C)^{\mathcal{I}} := (\forall r_1.\cdots \forall r_k.C)^{\mathcal{I}}$ where $w = r_1 \dots r_k \in {\mathsf{N_R}}^+$,
- $(\forall L.C)^{\mathcal{I}} := \bigcap_{w \in L} (\forall w.C)^{\mathcal{I}}$ and in particular $(\forall \emptyset.C)^{\mathcal{I}} = \Delta^{\mathcal{I}}$.

Subsumption (equivalence) between $\mathcal{FL}_{\text{reg}}$ concept descriptions w.r.t. an $\mathcal{FL}_0$ TBox is defined identically to the case of $\mathcal{FL}_0$: given $\mathcal{FL}_{\text{reg}}$ concept descriptions C, D and $\mathcal{FL}_0$ TBox $\mathcal{T}$, we say that C subsumes (is equivalent to) D w.r.t. $\mathcal{T}$, in symbols $C \sqsubseteq_{\mathcal{T}} D$ ($C \equiv_{\mathcal{T}} D$), if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ ($C^{\mathcal{I}} = D^{\mathcal{I}}$) for every model $\mathcal{I}$ of $\mathcal{T}$. The relevant decision problem is ExpTime-complete. The ExpTime upper bound follows from the known ExpTime upper bound for propositional dynamic logic (PDL) [Pra80; HS85] since $\mathcal{FL}_{\text{reg}}$ concept descriptions can be translated into PDL and GCIs can be internalized [Sch91]. The corresponding ExpTime lower bound already holds for subsumption between $\mathcal{FL}_0$ concept descriptions w.r.t. an $\mathcal{FL}_0$ TBox [BBL05], as we've seen in Section 4.1.

Given the above semantics of $\mathcal{FL}_{\text{reg}}$ concept descriptions, a normal form similar to 4.2 for $\mathcal{FL}_0$ can be obtained by applying the following equivalences (w.r.t. the empty TBox) as rewrite rules from left to right:

$$\forall L.(E \sqcap F) \equiv \forall L.E \sqcap \forall L.F, \quad \forall L.A \sqcap \forall L'.A \equiv \forall (L \cup L').A,$$
$$\forall L_1.\forall L_2.\dots \forall L_n.E \equiv \forall (L_1 \cdot L_2 \cdot \dots \cdot L_n).E,$$

where E, F are $\mathcal{FL}_{\text{reg}}$ concept descriptions, $A \in \mathsf{N_C}$, and $L, L_1, \dots, L_n$ are regular languages given as NFAs or regular expressions. In the case of NFAs, one needs to use their closure under union and concatenation in linear time[1] to obtain the new automata in polynomial time.

Using these rules, given $\mathcal{FL}_{\text{reg}}$ concept description C can be transformed in polynomial time into the following *normal form*:

$$C \equiv \forall K_1.A_1 \sqcap \dots \sqcap \forall K_k.A_k, \tag{8.1}$$

where $\mathsf{N_C} = \{A_1, \dots, A_k\}$ and $K_1, \dots, K_k$ are regular languages over $\mathsf{N_R}$ (given as regular expressions or NFAs). Concept names not occuring in C can be treated in the same way as before.

[1] See Lemma 3.5.

Later on, we will need to consider such normal forms, but where the regular languages are given as *deterministic* finite automata (DFAs). We call this normal form then *deterministic normal form (DNF)*. A DNF of an $\mathcal{FL}_{\text{reg}}$ concept description can obviously be obtained from its normal form by constructing DFAs from the regular expressions or NFAs, which may however result in DFAs that are exponentially larger than the original regular expressions or NFAs. In general, this normal form is not unique since different DFAs may accept the same language. One can make it unique (up to isomorphism of automata) by using minimal DFAs (see Lemma 3.4).

Value restriction sets with respect to an $\mathcal{FL}_0$ TBox can actually be defined for $\mathcal{FL}_{\text{reg}}$ concept descriptions in the exact same way as for $\mathcal{FL}_0$ concept descriptions, and a characterization of subsumption identical to Proposition 4.4 can be obtained.

Lemma 8.1. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C, D two $\mathcal{FL}_{reg}$ concept descriptions. Then, $C \sqsubseteq_{\mathcal{T}} D$ iff $\mathcal{L}_{\mathcal{T}}(D) \subseteq \mathcal{L}_{\mathcal{T}}(C)$, and $C \equiv_{\mathcal{T}} D$ iff $\mathcal{L}_{\mathcal{T}}(D) = \mathcal{L}_{\mathcal{T}}(C)$.*

Proof. First, assume that $C \sqsubseteq_{\mathcal{T}} D$, and consider $w \in \mathcal{L}_{\mathcal{T}}(D,A)$. Then we have $C \sqsubseteq_{\mathcal{T}} D \sqsubseteq_{\mathcal{T}} \forall w.A$. Transitivity of the subsumption relation implies $C \sqsubseteq_{\mathcal{T}} \forall w.A$, and thus $w \in \mathcal{L}_{\mathcal{T}}(C,A)$.

Second, assume that $C \not\sqsubseteq_{\mathcal{T}} D$, and C, D have the normal forms as depicted in (4.2). Then there is a $j, 1 \leq j \leq k$, such that $C \not\sqsubseteq_{\mathcal{T}} \forall L_j.A_j$. By the semantics of value restrictions in $\mathcal{FL}_{\text{reg}}$, this implies that there is a word $w \in L_j$ such that $C \not\sqsubseteq_{\mathcal{T}} \forall w.A_j$. Consequently, we have $w \in \mathcal{L}_{\mathcal{T}}(D,A_j) \setminus \mathcal{L}_{\mathcal{T}}(C,A_j)$, which implies $\mathcal{L}_{\mathcal{T}}(D) \not\subseteq \mathcal{L}_{\mathcal{T}}(C)$. □

Matching

We will now extend $\mathcal{FL}_0$ matching problems to allow for concept variables to be replaced by $\mathcal{FL}_{\text{reg}}$ concept descriptions. Other than this, we consider $\mathcal{FL}_0$ concept description, concept pattern, and TBox over $\mathsf{N_C}, \mathsf{N_R}, \mathsf{N_X}$ as input for the problem.

An *$\mathcal{FL}_{reg}$ substitution* σ is a mapping assigning $\mathcal{FL}_{\text{reg}}$ concept descriptions $\sigma(X)$ to variables $X \in \mathsf{N_X}$. The application of such a substitution σ to concept patterns is inductively defined as usual.

Definition 8.2 (Matching). *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox, C an $\mathcal{FL}_0$ concept description, and D an $\mathcal{FL}_0$ concept pattern. The $\mathcal{FL}_{reg}$ substitution σ is an* $\mathcal{FL}_{\text{reg}}$ matcher *of the matching problem $C \equiv^?_{\mathcal{T}} D$ w.r.t. $\mathcal{T}$ if $C \equiv_{\mathcal{T}} \sigma(D)$. If this matcher is an $\mathcal{FL}_0$ substitution, as before, we call it an* $\mathcal{FL}_0$ matcher. *An* $\mathcal{FL}_0$ matching problem *$C \equiv^?_{\mathcal{T}} D$ asks whether given such C, D and $\mathcal{T}$ there exists a matcher w.r.t. $\mathcal{T}$ or not.* ♢

When it matters whether the matcher is an $\mathcal{FL}_0$ or an $\mathcal{FL}_{\text{reg}}$ substitution, we will explicitly mention that. Otherwise, one can assume that we refer to the more general case of $\mathcal{FL}_{\text{reg}}$.

Example 8.3. *Let C and D respectively be the following $\mathcal{FL}_0$ concept description and $\mathcal{FL}_0$ concept pattern (in normal form):*

$$C := \forall\{r,s\}.A \sqcap \forall\{s\}.B, \quad D := \forall\{rr\}.A \sqcap \forall\{r,s\}.X_1 \sqcap \forall\{s\}.X_2.$$

It is easy to see that $\mathcal{L}_{\emptyset}(C,A) = \{r,s\}$. Since D has the conjunct $\forall\{rr\}.A$, we know that the word rr belongs to $\mathcal{L}_{\emptyset}(\sigma(D),A)$ for all substitutions σ. By the characterization of subsumption

given in Lemma 8.1, this implies that the matching problem $C \equiv^? D$ has no matcher w.r.t. the empty TBox.

However, if we consider $C \equiv^?_{\mathcal{T}} D$ w.r.t. TBox $\mathcal{T} := \{A \sqsubseteq \forall r.A, \forall s.B \sqsubseteq A\}$, there actually exists an $\mathcal{FL}_0$ matcher for this problem. Notice that the GCI $\forall s.B \sqsubseteq A$ implies that $C \sqsubseteq_{\mathcal{T}} A$. Moreover, $A \sqsubseteq \forall r.A$ yields that $A \sqsubseteq_{\mathcal{T}} \forall w.A$ for all $w \in \{r\}^*$. Hence, it follows that $\mathcal{L}_{\mathcal{T}}(C,A) = \{s,\varepsilon\}\{r\}^*$ and $\mathcal{L}_{\mathcal{T}}(C,B) = \{s\}$. By setting $\sigma(X_1) := A$ and $\sigma(X_2) := B$, we have that $\mathcal{L}_{\mathcal{T}}(C,A) = \mathcal{L}_{\mathcal{T}}(\sigma(D),A)$ and $\mathcal{L}_{\mathcal{T}}(C,B) = \mathcal{L}_{\mathcal{T}}(\sigma(D),B)$. Thus, Lemma 8.1 implies that σ is a matcher for $C \equiv^?_{\mathcal{T}} D$. ♢

Recall the polynomial-time algorithm for deciding whether an $\mathcal{FL}_0$ matching problem has an $\mathcal{FL}_0$ matcher w.r.t. the empty TBox that was introduced in [BN01] and we presented in Section 7.1. Essentially, it is based on the observation that such a problem has a matcher iff a certain candidate substitution is a matcher. The algorithm then computes this candidate substitution and checks whether it is indeed a matcher. Basically, our matching algorithm proceeds in the same way, but we need to overcome two problems. First, the candidate substitution is an $\mathcal{FL}_{\text{reg}}$ substitution rather than an $\mathcal{FL}_0$ substitution. Thus, we actually check whether the problem has an $\mathcal{FL}_{\text{reg}}$ matcher. However, we then show that the existence of an $\mathcal{FL}_{\text{reg}}$ matcher also implies the existence of an $\mathcal{FL}_0$ matcher. Second, the candidate matcher may already be of exponential size. Thus, if we just use the result that subsumption in $\mathcal{FL}_{\text{reg}}$ w.r.t. an $\mathcal{FL}_0$ TBox is in ExpTime, we obtain a doubly-exponential upper bound for the overall complexity of checking whether the candidate substitution really is a matcher. In order to bring this upper bound down to ExpTime, we need a more fine-grained analysis of the complexity of the subsumption problem, which we provide in the next section.

8.2 Subsumption in $\mathcal{FL}_{\text{reg}}$ w.r.t. an $\mathcal{FL}_0$ TBox

Given $\mathcal{FL}_{\text{reg}}$ concept descriptions C, D and an $\mathcal{FL}_0$ TBox $\mathcal{T}$, we are interested in the complexity of deciding whether $C \sqsubseteq_{\mathcal{T}} D$ holds or not. Basically, we will use the characterization of subsumption given in Lemma 8.1 to obtain a subsumption algorithm. However, it turns out that a model-theoretic variant of this characterization is more appropriate to achieve a fine-grained complexity analysis that distinguishes between the size of C, D and the size of $\mathcal{T}$. For subsumption between $\mathcal{FL}_0$ concept descriptions w.r.t. an $\mathcal{FL}_0$ TBox $\mathcal{T}$, such a semantic characterization has been introduced in [Pen15; BFP18].

Definition 8.4. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C an $\mathcal{FL}_{reg}$ concept description. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is called a* functional interpretation *if $\Delta^{\mathcal{I}} = \mathsf{N_R}^*$ and $r^{\mathcal{I}} := \{(u, ur) \mid u \in \mathsf{N_R}^*\}$ for all $r \in \mathsf{N_R}$. The functional interpretation $\mathcal{I}$ is called a*

- functional model of C *if $\varepsilon \in C^{\mathcal{I}}$,*
- functional model of $\mathcal{T}$ *if $\mathcal{I}$ is a model of $\mathcal{T}$,*
- functional model of C w.r.t. $\mathcal{T}$ *if $\varepsilon \in C^{\mathcal{I}}$ and $\mathcal{I}$ is a model of $\mathcal{T}$.* ♢

Calling such interpretations *functional* is justified by the fact that they interpret roles as (total) functions: for every $u \in \mathsf{N_R}^*$ and every $r \in \mathsf{N_R}$, the word ur is the unique r-successor

of u. As an immediate consequence of this functional interpretation of roles, we have for all $A \in \mathsf{N_C}$ and $u, w \in {\mathsf{N_R}}^*$:

$$w \in (\forall u.A)^{\mathcal{I}} \text{ iff } wu \in A^{\mathcal{I}}.$$

We define inclusion and intersection of functional interpretations as follows:

- $\mathcal{I} \subseteq \mathcal{J}$ if $A^{\mathcal{I}} \subseteq A^{\mathcal{J}}$ for all $A \in \mathsf{N_C}$;
- $\mathcal{I} \cap \mathcal{J}$ is the unique functional interpretation that satisfies $A^{\mathcal{I}\cap\mathcal{J}} = A^{\mathcal{I}} \cap A^{\mathcal{J}}$ for all $A \in \mathsf{N_C}$.

It is easy to see that the above classes of functional models are closed under intersection, i.e., if $\mathcal{I}$ and $\mathcal{J}$ are both functional models of C w.r.t. $\mathcal{T}$ (and likewise of C, or of $\mathcal{T}$), then so is their intersection $\mathcal{I} \cap \mathcal{J}$. This actually not only holds for binary intersection, but also for arbitrary intersection of functional models. In particular, this implies that there must exist a *least functional model* of C w.r.t. $\mathcal{T}$, i.e., a functional model $\mathcal{J}$ of C w.r.t. $\mathcal{T}$ such that $\mathcal{J} \subseteq \mathcal{I}$ holds for all functional models $\mathcal{I}$ of C w.r.t. $\mathcal{T}$. There is a close connection between the least functional models and the value restriction sets introduced in the previous section.

Proposition 8.5. *Given an $\mathcal{FL}_{reg}$ concept description C and an $\mathcal{FL}_0$ TBox $\mathcal{T}$, let $\mathcal{I}_{C,\mathcal{T}} = ({\mathsf{N_R}}^*, \cdot^{\mathcal{I}_{C,\mathcal{T}}})$ be the functional interpretation satisfying*

$$A^{\mathcal{I}_{C,\mathcal{T}}} = \{w \in {\mathsf{N_R}}^* \mid w \in \mathcal{L}_{\mathcal{T}}(C, A)\} \text{ for all } A \in \mathsf{N_C}.$$

Then, $\mathcal{I}_{C,\mathcal{T}}$ is the least functional model of C w.r.t. $\mathcal{T}$. ♢

The proof of this proposition is identical to the one given in [BFP18] for the case where C is an $\mathcal{FL}_0$ concept description. Combining this result with Lemma 8.1 we can immediately conclude the following.

Corollary 8.6. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C, D $\mathcal{FL}_{reg}$ concept descriptions. Then $C \sqsubseteq_{\mathcal{T}} D$ iff $\mathcal{I}_{D,\mathcal{T}} \subseteq \mathcal{I}_{C,\mathcal{T}}$.*

In order to test the condition $\mathcal{I}_{D,\mathcal{T}} \subseteq \mathcal{I}_{C,\mathcal{T}}$, we want to represent these least functional models using tree automata.

Functional interpretations can be represented as $\mathsf{N_R}$-trees with labels from the set $2^{\mathsf{N_C}}$. More precisely, given a functional interpretation $\mathcal{I}$, the $2^{\mathsf{N_C}}$-labeled $\mathsf{N_R}$-tree $t_{\mathcal{I}}$ corresponding to $\mathcal{I}$ is defined as $t_{\mathcal{I}}(w) := \{A \in \mathsf{N_C} \mid w \in A^{\mathcal{I}}\}$.[2] Conversely, any tree $t \in T^{\omega}_{\mathsf{N_R},2^{\mathsf{N_C}}}$ induces a functional interpretation $\mathcal{I}_t$ where $A^{\mathcal{I}_t} := \{w \in {\mathsf{N_R}}^* \mid A \in t(w)\}$ for every $A \in \mathsf{N_C}$. These two mappings are bijections that are inverse to each other. In the following, we will not always distinguish between a functional interpretation and its tree representation. For example, we will say that an automaton recognizes a functional interpretation $\mathcal{I}$ rather than use the (more exact) expression that it recognizes the tree representation $t_{\mathcal{I}}$ of $\mathcal{I}$.

In [BFP18] it is shown how to construct an LTA $\mathcal{A}_{C,\mathcal{T}}$ that recognizes the functional models of an $\mathcal{FL}_0$ concept description C w.r.t. an $\mathcal{FL}_0$ TBox $\mathcal{T}$. If we set $C = \top$, then this automaton actually recognizes the functional models of $\mathcal{T}$. Thus, the construction and the results in [BFP18] (in particular, Definition 8 and Lemma 9 of [BFP18]) provide us with the following results.

[2] Recall our discussion after Theorem 4.6 on the equivalence between the label sets $\{\}^k$ and $2^{\mathsf{N_C}}$.

Proposition 8.7. *Given an $\mathcal{FL}_0$ TBox $\mathcal{T}$, we can construct an LTA $\mathcal{A}_\mathcal{T}$ such that*

$$\mathcal{L}(\mathcal{A}_\mathcal{T}) = \{t_\mathcal{I} \mid \mathcal{I} \text{ is a functional model of } \mathcal{T}\}.$$

The size of $\mathcal{A}_\mathcal{T}$ is exponential in the size of $\mathcal{T}$, and it can be constructed in exponential time. ♢

Now, consider an $\mathcal{FL}_{\text{reg}}$ concept description $C = \forall L_1.A_1 \sqcap \cdots \sqcap \forall L_k.A_k$ in deterministic normal form, and let $\mathcal{A}_1, \dots, \mathcal{A}_k$ be the DFAs recognizing the languages $L_1, \dots, L_k$. Similarly to Lemma 3.21, we can construct an LTA that recognizes all the functional models of C (and not only the tree representing $\mathcal{L}_\mathcal{T}(C)$, as the lemma would give us). In the following, we assume that these DFAs are of the form $\mathcal{A}_i = (Q_i, \mathsf{N_R}, q_i^0, \delta_i, F_i)$ where Q_i is the set of states, $\mathsf{N_R}$ the alphabet, q_i^0 the initial state, $\delta_i : Q_i \times \mathsf{N_R} \to Q_i$ the transition function, and $F_i \subseteq Q_i$ the set of final states. We can use these DFAs to construct an LTA $\mathcal{A}_C$ that recognizes the functional models of C. To be more precise, we define $\mathcal{A}_C := (P, \Sigma, L, \Delta, \{p^0\})$, where $P := Q_1 \times \cdots \times Q_k$, $\Sigma := \mathsf{N_R} = \{r_1, \dots, r_n\}$, $L := 2^{\{A_1, \dots, A_k\}}$, $p^0 := (q_1^0, \dots, q_k^0)$, and

$$\begin{aligned}\Delta := \{(p, \ell, p_1, \dots, p_n) \mid\ & p = (q_1, \dots, q_k) \in P,\ \ \{A_i \mid 1 \le i \le k, q_i \in F_i\} \subseteq \ell, \\ & p_i = (\delta_1(q_1, r_i), \dots, \delta_k(q_k, r_i)) \text{ for } i = 1, \dots, n \quad \}\end{aligned}$$

Lemma 8.8. $\mathcal{L}(\mathcal{A}_C) = \{t_\mathcal{I} \mid \mathcal{I} \text{ is a functional model of } C\}$.

Proof. Since the automata $\mathcal{A}_i$ are deterministic, the automaton $\mathcal{A}_C$ has at most one run ρ, where $\rho(w) = (\delta_1(q_1^0, w), \dots, \delta_n(q_k^0, w))$. For a given tree t, ρ is indeed a run on the tree t iff the following holds for all $w \in \mathsf{N_R}^*$: $t(w)$ contains all concept names A_i with $\delta_i(q_i^0, w) \in F_i$, i.e., it contains all A_i with $w \in L_i$. Consequently, the tree t is accepted by $\mathcal{A}_C$ iff $w \in A_i^{\mathcal{I}_t}$ holds for all $w \in L_i$. Since $w \in A_i^{\mathcal{I}_t}$ is equivalent to $\varepsilon \in (\forall w.A_i)^{\mathcal{I}_t}$, this shows that $\mathcal{A}_C$ accepts exactly the tree-representations of functional models of C. □

Note that the size of $\mathcal{A}_C$ is bounded by $h(m^k)$, where m is the maximal size of the automata $\mathcal{A}_1, \dots, \mathcal{A}_k$, k is the number of concept names occurring in C, and h is a polynomial. In order to obtain an automaton that recognizes all functional models of C w.r.t. $\mathcal{T}$, we can apply the standard product construction to obtain an automaton recognizing $\mathcal{L}(\mathcal{A}_C) \cap \mathcal{L}(\mathcal{A}_\mathcal{T}) = \{t_\mathcal{I} \mid \mathcal{I} \text{ is a functional model of } C \text{ w.r.t. } \mathcal{T}\}$.

Proposition 8.9. *Given an $\mathcal{FL}_{reg}$ concept description C in DNF and an $\mathcal{FL}_0$ TBox $\mathcal{T}$, we can construct an LTA $\mathcal{A}_{C,\mathcal{T}}$ such that $\mathcal{L}(\mathcal{A}_{C,\mathcal{T}}) = \{t_\mathcal{I} \mid \mathcal{I} \text{ is a functional model of } C \text{ w.r.t. } \mathcal{T}\}$. If m is the maximal size of the DFAs used to represent regular languages in C, k is the number of concept names occurring in C or $\mathcal{T}$, and τ is the size of $\mathcal{T}$, then the size of $\mathcal{A}_{C,\mathcal{T}}$ is bounded by $2^{h_1(\tau)} \cdot h_2(m^k)$ for polynomials h_1, h_2.* ♢

Just as in the case of an $\mathcal{FL}_0$ concept description C, the automaton $\mathcal{A}_{C,\mathcal{T}}$ can be transformed into an LTA that accepts exactly the least functional model of C w.r.t. $\mathcal{T}$. This transformation removes states and transitions (see Definition 10 and Theorem 11 in [BFP18]).

Proposition 8.10. *Given an $\mathcal{FL}_{reg}$ concept description C in DNF and an $\mathcal{FL}_0$ TBox $\mathcal{T}$, we can construct an LTA $\widehat{\mathcal{A}}_{C,\mathcal{T}}$ such that $\mathcal{L}(\widehat{\mathcal{A}}_{C,\mathcal{T}}) = \{t_{\mathcal{I}_{C,\mathcal{T}}}\}$. The size of $\widehat{\mathcal{A}}_{C,\mathcal{T}}$ is bounded by the size of $\mathcal{A}_{C,\mathcal{T}}$.* ♢

Now, let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C, D $\mathcal{FL}_{\text{reg}}$ concept descriptions in DNF. According to Corollary 8.6, we have $C \sqsubseteq_{\mathcal{T}} D$ iff $\mathcal{I}_{D,\mathcal{T}} \subseteq \mathcal{I}_{C,\mathcal{T}}$. As shown in [BFP18], the latter condition can be reduced to the emptiness problem for an LTA that is obtained from $\mathcal{A}_{C,\mathcal{T}}$ and $\mathcal{A}_{D,\mathcal{T}}$ using an appropriate product construction (see the construction above Corollary 12 in [BFP18]). Since the emptiness problem for LTAs can be decided in linear time, this yields the following fine-grained complexity result for subsumption.

Theorem 8.11. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C, D $\mathcal{FL}_{reg}$ concept descriptions in DNF. Then subsumption between C and D w.r.t. $\mathcal{T}$ can be decided in time at most $2^{h_1(\tau)} \cdot h_2(m^k)$ where m is the maximal size of the DFAs used to represent regular languages in C, D, k is the number of concept names occurring in C, D or $\mathcal{T}$, τ is the size of $\mathcal{T}$, and h_1, h_2 are polynomials.*

If we start with arbitrary $\mathcal{FL}_{\text{reg}}$ concept descriptions C, D, then we can construct equivalent normal forms in polynomial time without changing the set of concept names occurring in these concept descriptions. Transforming the regular expressions or NFAs representing the regular languages in these normal forms into equivalent deterministic automata may produce DFAs whose size is exponential in the size of C, D. Thus, the maximal size m of the DFAs occurring in the DNFs of C, D is bounded by 2^s where s is the combined size of C, D. Thus, $h_2(m^k) = h_2((2^s)^k) = h_2(2^{s \cdot k})$ is still single-exponential in the size of C, D.

Corollary 8.12. *Let C, D be $\mathcal{FL}_{reg}$ concept descriptions, and $\mathcal{T}$ an $\mathcal{FL}_0$ TBox. Then subsumption between C and D w.r.t. $\mathcal{T}$ can be decided in time exponential in the combined size of C, D, and $\mathcal{T}$.*

8.3 The complexity of matching in $\mathcal{FL}_0$ w.r.t. TBoxes

Since subsumption in $\mathcal{FL}_0$ w.r.t. a TBox is ExpTime-complete [BBL05], matching w.r.t. a TBox is ExpTime-hard. In fact, we have $E \sqsubseteq_{\mathcal{T}} F$ iff the matching problem $C \equiv^? D$ has a matcher w.r.t. $\mathcal{T}$, where $C = E \sqcap F$ and $D = E$ is a variable-free pattern. This hardness result is, of course, independent of whether we are looking for a matcher in $\mathcal{FL}_{\text{reg}}$ or in $\mathcal{FL}_0$. In this section, we will show the corresponding upper bounds, first for the existence of an $\mathcal{FL}_{\text{reg}}$ matcher, and then for $\mathcal{FL}_0$ matchers.

Deciding the existence of an $\mathcal{FL}_{\text{reg}}$-matcher

By applying the normalization rules described above to the $\mathcal{FL}_0$ concept pattern D, a given $\mathcal{FL}_0$ matching problem $C \equiv^?_{\mathcal{T}} D$ can be equivalently stated as an equation of the form:

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_m.X_m \tag{8.2}$$

where E is an $\mathcal{FL}_0$ concept description, $L_1, \ldots, L_m$ are finite languages over $\mathsf{N_R}$ (given by the enumeration of their elements), and $X_1, \ldots, X_m$ are the concept variables occurring in D. Generalizing the approach for matching in $\mathcal{FL}_0$ without a TBox [BN01], we now show that an equation of the form (8.2) has an $\mathcal{FL}_{\text{reg}}$ matcher iff a certain candidate substitution is a matcher.

Let σ be an $\mathcal{FL}_{\text{reg}}$ matcher of (8.2) such that $\sigma(X_i)$ is in normal form, and assume that $\forall L_{i,j}.A_j$ is the conjunct for the concept name A_j in $\sigma(X_i)$. Then, after applying the substitution

σ to the right-hand side of the equation (8.2), the value restriction $\forall L_i{\cdot}L_{i,j}.A_j$ is a conjunct on the right-hand side, and thus subsumes C. Lemma 8.1 thus implies that $L_i{\cdot}L_{i,j} \subseteq \mathcal{L}_\mathcal{T}(C,A_j)$. Now, assume that $v \in L_{i,j}$. Then we know that $uv \in \mathcal{L}_\mathcal{T}(C,A_j)$ must hold for every $u \in L_i$, i.e., $v \in u^{-1}\mathcal{L}_\mathcal{T}(C,A_j)$ for all $u \in L_i$. This shows that $L_{i,j} \subseteq \bigcap_{u\in L_i} u^{-1}\mathcal{L}_\mathcal{T}(C,A_j)$. At first sight, this does not help us in our search for matchers since the languages $\mathcal{L}_\mathcal{T}(C,A_j)$ are infinite, and there are thus possibly infinitely many choices for such subsets to consider. However, we can show that we can restrict our attention to the maximal such sets. To be more precise, we define for $i = 1,\dots,m$ and $j = 1,\dots,k$ the languages

$$\widehat{L}_{i,j} := \bigcap_{u\in L_i} u^{-1}\mathcal{L}_\mathcal{T}(C,A_j).$$

Since the class of regular languages is closed under building left-quotients and finite intersections, and the languages $\mathcal{L}_\mathcal{T}(C,A_j)$ are regular [Pen15, Theorem 5.21], the languages $\widehat{L}_{i,j}$ are also regular. Thus, we can use them within $\mathcal{FL}_{\text{reg}}$ concept descriptions. Consequently, if we define the *candidate substitution* $\widehat{\sigma}$ as

$$\widehat{\sigma}(X_i) := \forall\widehat{L}_{i,1}.A_1 \sqcap \dots \sqcap \forall\widehat{L}_{i,k}.A_k \text{ for } i = 1,\dots,m,$$

then $\widehat{\sigma}$ is a well-defined $\mathcal{FL}_{\text{reg}}$ substitution.

Lemma 8.13. *The equation (8.2) has an $\mathcal{FL}_{reg}$ matcher iff the candidate substitution $\widehat{\sigma}$ is a matcher of (8.2).*

Proof. Since $\widehat{\sigma}$ is an $\mathcal{FL}_{\text{reg}}$ substitution, the if-direction of the proof is trivial. To show the other direction, assume that equation (8.2) has an $\mathcal{FL}_{\text{reg}}$ matcher σ, i.e., σ is an $\mathcal{FL}_{\text{reg}}$ substitution such that $C \equiv_\mathcal{T} E \sqcap \forall L_1.\sigma(X_1) \sqcap \dots \sqcap \forall L_m.\sigma(X_m)$. This implies that $C \sqsubseteq_\mathcal{T} E$. Moreover, the construction of $\widehat{\sigma}$ implies that $C \sqsubseteq_\mathcal{T} \forall L_i.\widehat{\sigma}(X_i)$ for all $i, 1 \le i \le m$ since $L_i{\cdot}\widehat{L}_{i,j} \subseteq \mathcal{L}_\mathcal{T}(C,A_j)$ holds for all $j, 1 \le j \le k$. Consequently, we have $C \sqsubseteq_\mathcal{T} E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \dots \sqcap \forall L_m.\widehat{\sigma}(X_m)$.

To see the opposite direction, assume that $\sigma(X_i) = \forall L_{i,1}.A_1 \sqcap \dots \sqcap \forall L_{i,k}.A_k$ for $i = 1,\dots,m$. As argued above, the fact that σ is an $\mathcal{FL}_{\text{reg}}$ matcher of (8.2) implies that $L_{i,j} \subseteq \widehat{L}_{i,j}$ holds for all $i, 1 \le i \le m$ and $j, 1 \le j \le k$. Consequently, we have $\widehat{\sigma}(X_i) \sqsubseteq \sigma(X_i)$ for all $i, 1 \le i \le m$, which yields $E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \dots \sqcap \forall L_m.\widehat{\sigma}(X_m) \sqsubseteq_\mathcal{T} E \sqcap \forall L_1.\sigma(X_1) \sqcap \dots \sqcap \forall L_k.\sigma(X_m) \equiv_\mathcal{T} C$. Thus, we can conclude that $\widehat{\sigma}$ is a matcher of (8.2). □

This lemma reduces deciding whether (8.2) has an $\mathcal{FL}_{\text{reg}}$ matcher to deciding whether $\widehat{\sigma}$ is a matcher of (8.2). The latter can be checked as follows.

Lemma 8.14. *The candidate substitution $\widehat{\sigma}$ is a matcher of (8.2) iff*

$$\textit{1. } C \sqsubseteq_\mathcal{T} E, \quad \textit{and} \quad \textit{2. } E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \dots \sqcap \forall L_m.\widehat{\sigma}(X_m) \sqsubseteq_\mathcal{T} C.$$

The first condition requires testing subsumption of $\mathcal{FL}_0$ concept descriptions w.r.t. an $\mathcal{FL}_0$ TBox, which can be performed in exponential time. The second condition requires testing subsumption of $\mathcal{FL}_{\text{reg}}$ concept descriptions w.r.t. an $\mathcal{FL}_0$ TBox. However, we cannot directly apply Corollary 8.12 since the size of these concept descriptions need not be polynomial in the combined size of C, D, and $\mathcal{T}$. To show that this test can also be performed in exponential time, we must use the more fine-grained complexity result for subsumption of $\mathcal{FL}_{\text{reg}}$ concept

descriptions in DNF w.r.t. an $\mathcal{FL}_0$ TBox of Theorem 8.11. To obtain the desired ExpTime upper bound, we thus need to show that there are DFAs recognizing the regular languages in the normal form of $E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \ldots \sqcap \forall L_m.\widehat{\sigma}(X_m)$ that are of size at most exponential in the combined size of C, D, and $\mathcal{T}$. Assume the normal form of E is $\forall K_1.A_1 \sqcap \ldots \sqcap \forall K_k.A_k$, for finite languages $K_1, \ldots, K_k$. Then the normal form of $E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \ldots \sqcap \forall L_m.\widehat{\sigma}(X_m)$ is $\forall M_1.A_1 \sqcap \ldots \sqcap \forall M_k.A_k$, where

$$M_j = K_j \cup L_1 \cdot \widehat{L}_{1,j} \cup \ldots \cup L_m \cdot \widehat{L}_{m,j} \text{ for } j = 1, \ldots, k.$$

Lemma 8.15. *For all $j, 1 \leq j \leq k$, there is a DFA recognizing M_j whose size is at most exponential in the combined size of C, D, and $\mathcal{T}$.*

Proof. We start the proof by constructing DFAs of appropriate size for the languages $\widehat{L}_{i,j} = \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$. As shown in [Pen15], the languages $\mathcal{L}_{\mathcal{T}}(C, A_j)$ are recognized by DFAs $\mathcal{A}_j$ whose sizes are at most exponential in the combined size of C and $\mathcal{T}$. For every $j, 1 \leq j \leq k$ and $u \in L_i$, a DFA $\mathcal{A}_{j,u}$ for $u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$ is obtained from $\mathcal{A}_j$ by taking as new initial state the state reached with u from the initial state of $\mathcal{A}_j$. The intersection $\bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$ can then be realized by building the product automaton $\mathcal{P}_{i,j}$ of the automata $\mathcal{A}_{j,u}$ for all $u \in L_i$. The size of $\mathcal{P}_{i,j}$ is exponential in $|L_i|$ times the combined size of C and $\mathcal{T}$, and thus exponential in the combined size of C, D, and $\mathcal{T}$.

Second, let us consider the concatenation $L_i \cdot \widehat{L}_{i,j}$. For every $u \in L_i$ we can construct a DFA $\mathcal{P}^u_{i,j}$ for $\{u\} \cdot \widehat{L}_{i,j}$ by adding $|u|$ many states "before" the initial state of $\mathcal{P}_{i,j}$. A DFA $\widehat{\mathcal{P}}_{i,j}$ for the union $\bigcup_{u \in L_i} \{u\} \cdot \widehat{L}_{i,j} = L_i \cdot \widehat{L}_{i,j}$ can then again be obtained by a product construction. The exponent $|L_i|$ in the size of $\widehat{\mathcal{P}}_{i,j}$ caused by this product construction becomes a factor in the overall exponent and it is bounded by the size of the pattern D. Consequently, the sizes of the DFAs $\widehat{\mathcal{P}}_{i,j}$ recognizing $L_i \cdot \widehat{L}_{i,j}$ are again exponential in the combined size of C, D, and $\mathcal{T}$.

Finally, the union $K_j \cup L_1 \cdot \widehat{L}_{1,j} \cup \ldots \cup L_m \cdot \widehat{L}_{m,j}$ can again be realized by a product construction, where the exponent $m + 1$ caused by this construction again becomes a factor in the overall exponent and is bounded by the size of the pattern D. □

Together with Theorem 8.11, this lemma yields the desired ExpTime upper bound.

Theorem 8.16. *The problem of deciding whether an $\mathcal{FL}_0$ matching problem w.r.t. an $\mathcal{FL}_0$ TBox has an $\mathcal{FL}_{reg}$ matcher or not is* ExpTime*-complete.*

We illustrate our ExpTime decision procedure using the matching problem of Example 8.3.

Example 8.17. *Consider the matching problem $C \equiv^?_{\mathcal{T}} D$, where*

$$C := \forall\{r,s\}.A \sqcap \forall\{s\}.B, \quad D := \forall\{rr\}.A \sqcap \forall\{r,s\}.X_1 \sqcap \forall\{s\}.X_2, \quad \mathcal{T} := \{A \sqsubseteq \forall r.A, \forall s.B \sqsubseteq A\}.$$

We have seen in Example 8.3 that $\mathcal{L}_{\mathcal{T}}(C, A) = \{s, \varepsilon\}\{r\}^*$ and $\mathcal{L}_{\mathcal{T}}(C, B) = \{s\}$. Since $r^{-1}\mathcal{L}_{\mathcal{T}}(C, A) \cap s^{-1}\mathcal{L}_{\mathcal{T}}(C, A) = \{r\}^* \cap \{r\}^* = \{r\}^*$ and $r^{-1}\mathcal{L}_{\mathcal{T}}(C, B) \cap s^{-1}\mathcal{L}_{\mathcal{T}}(C, B) = \emptyset \cap \{\varepsilon\} = \emptyset$, the value of the candidate substitution for X_1 is $\widehat{\sigma}(X_1) := \forall r^*.A$. Regarding X_2, we have that $s^{-1}\mathcal{L}_{\mathcal{T}}(C, A) = \{r\}^*$ and $s^{-1}\mathcal{L}_{\mathcal{T}}(C, B) = \{\varepsilon\}$, which yields $\widehat{\sigma}(X_2) := \forall r^*.A \sqcap B$. With basic calculations we can verify that $\widehat{\sigma}$ is in fact a matcher. This substitution is an $\mathcal{FL}_{reg}$ matcher, but it is not an $\mathcal{FL}_0$ substitution. Nevertheless, the matching problem has $\mathcal{FL}_0$ matchers, as we have seen in Example 8.3. Next, we will prove that this is not simply a coincidence. ◇

Deciding the existence of an $\mathcal{FL}_0$-matcher

We will show that a matching problem of the form (8.2) has an $\mathcal{FL}_0$ matcher iff it has an $\mathcal{FL}_{\text{reg}}$ matcher. We have shown that any matcher σ of (8.2) with normal form

$$\sigma(X_i) := \forall L_{i,1}.A_1 \sqcap \ldots \sqcap \forall L_{i,k}.A_k \text{ for } i = 1, \ldots, m, \tag{8.3}$$

satisfies $L_{i,j} \subseteq \widehat{L}_{i,j}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, k$. Obviously, this substitution is an $\mathcal{FL}_0$ substitution iff the languages $L_{i,j}$ are finite.

Analogously to Lemma 8.14 we can thus show the following lemma characterizing the existence of $\mathcal{FL}_0$ matchers.

Lemma 8.18. *Equation (8.2) has an $\mathcal{FL}_0$ matcher iff*

1. $C \sqsubseteq_{\mathcal{T}} E$.
2. *There are finite languages $L_{i,j} \subseteq \widehat{L}_{i,j}$ such that $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} C$, where σ is defined as in (8.3).*

We claim that the second condition is satisfied if the candidate substitution $\widehat{\sigma}$ satisfies the corresponding condition. Before we can prove this implication, we need to introduce some more notation. A possibly negated $\mathcal{FL}_0$ *concept assertion* is of the form $C(a)$ or $\neg C(a)$ where C is an $\mathcal{FL}_0$ concept description and a is an individual name from a set $\mathsf{N_I}$ of such names. We extend the semantics of $\mathcal{FL}_0$ such that interpretations $\mathcal{I}$ assign elements $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to individual names $a \in \mathsf{N_I}$. Given a (finite or infinite) set M of such assertions, we say that M is *consistent* w.r.t. the $\mathcal{FL}_0$ TBox $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all positive concept assertions $C(a)$ in M and $a^{\mathcal{I}} \notin C^{\mathcal{I}}$ holds for all negative concept assertions $\neg C(a)$ in M.

Lemma 8.19. *If $E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \ldots \sqcap \forall L_m.\widehat{\sigma}(X_m) \sqsubseteq_{\mathcal{T}} C$, then Condition 2 in Lemma 8.18 is satisfied.*

Proof. It is easy to see that $E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \ldots \sqcap \forall L_m.\widehat{\sigma}(X_m) \sqsubseteq_{\mathcal{T}} C$ holds in $\mathcal{FL}_{\text{reg}}$ iff the following (possibly infinite) set of assertions is inconsistent w.r.t. the $\mathcal{FL}_0$ TBox $\mathcal{T}$:

$$\Big\{E(a), \neg C(a)\Big\} \cup \bigcup_{i=1}^{m} \bigcup_{j=1}^{k} \Big\{(\forall uv.A_j)(a) \mid u \in L_i \wedge v \in \widehat{L}_{i,j}\Big\}.$$

Since $\mathcal{FL}_0$ TBoxes and possibly negated $\mathcal{FL}_0$ concept assertions can clearly be translated into sentences of first-order logic (FOL), compactness of FOL ([Hod97]) implies that there is a *finite* set $\Gamma \subseteq \bigcup_{i=1}^{m} \bigcup_{j=1}^{k} \Big\{(\forall uv.A_j)(a) \mid u \in L_i \wedge v \in \widehat{L}_{i,j}\Big\}$ such that $\{E(a), \neg C(a)\} \cup \Gamma$ is inconsistent w.r.t. $\mathcal{T}$. We use Γ to define finite subsets $L_{i,j}$ of $\widehat{L}_{i,j}$:

$$L_{i,j} := \Big\{v \in \widehat{L}_{i,j} \mid \text{ there is } u \in L_i \text{ such that } (\forall uv.A_j)(a) \in \Gamma\Big\}.$$

Then Γ is a subset of the set $\Gamma' := \bigcup_{i=1}^{m} \bigcup_{j=1}^{k} \Big\{\forall uv.A(a) \mid u \in L_i, v \in L_{i,j}\Big\}$, which implies that $\{E(a), \neg C(a)\} \cup \Gamma'$ is also inconsistent w.r.t. $\mathcal{T}$. By defining σ as in (8.3), this in turn implies that $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} C$. □

We are now ready to show the following equivalence.

Theorem 8.20. *An $\mathcal{FL}_0$ matching problem has an $\mathcal{FL}_{reg}$ matcher iff it has an $\mathcal{FL}_0$ matcher.*

Proof. Clearly, an $\mathcal{FL}_0$ matcher is also an $\mathcal{FL}_{\text{reg}}$ matcher. Conversely, assume that the matching problem is of the form (8.2) and that it has an $\mathcal{FL}_{\text{reg}}$ matcher. Then the two conditions in Lemma 8.14 are satisfied. The first condition coincides with the first condition in Lemma 8.18 and, according to Lemma 8.19, the second condition implies the second condition in Lemma 8.18. Thus, Lemma 8.18 yields the existence of an $\mathcal{FL}_0$ matcher. □

As an immediate consequence of Theorem 8.16, we thus obtain the following complexity result.

Corollary 8.21. *The problem of deciding whether an $\mathcal{FL}_0$ matching problem w.r.t. an $\mathcal{FL}_0$ TBox has an $\mathcal{FL}_0$ matcher or not is* ExpTime*-complete.*

8.4 Subsumption and matching w.r.t. forward TBoxes

Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox. We assume in the following that the GCIs in $\mathcal{T}$ are of the form

$$\forall v_1.A_1 \sqcap \cdots \sqcap \forall v_s.A_s \sqsubseteq \forall v.A, \tag{8.4}$$

where $v_1, \ldots, v_s, v \in \mathsf{N_R}^*$ and $A_1, \ldots, A_s, A \in \mathsf{N_C}$. This is without loss of generality since (i) any $\mathcal{FL}_0$ concept description is equivalent to a conjunction of value restrictions of the form $\forall w.B$, and (ii) $C \sqsubseteq D \sqcap E$ iff $C \sqsubseteq D$ and $C \sqsubseteq E$.

Definition 8.22. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox. Then $\mathcal{T}$ is called a* forward TBox *if all its GCIs are of the form (8.4) where $|v_i| \leq |v|$ for all $i, 1 \leq i \leq s$.* ♢

For example, the GCI $A \sqsubseteq \forall r.A$ can be an element of a forward TBox, but the GCI $\forall r.B \sqsubseteq A$ cannot. We show in the following that restricting to forward TBoxes lowers the complexity of subsumption and matching from ExpTime to PSpace. Actually, the main contribution of this section is developing the PSpace subsumption algorithm. The PSpace matching algorithm can then be obtained from it by a simple modification.

Subsumption in $\mathcal{FL}_0$ w.r.t. forward TBoxes

We assume in the following that all $\mathcal{FL}_0$ concept descriptions are conjunctions of value restrictions of the form $\forall w.B$ for $w \in \mathsf{N_R}^*$ and $B \in \mathsf{N_C}$. Given such a concept description D, we denote with $\widehat{D}$ the set of these value restrictions. For example, if $D = \forall r r.A \sqcap \forall s.A \sqcap \forall s.B$, then $\widehat{D} = \{\forall r r.A, \forall s.A, \forall s.B\}$.

For the same reason that GCIs can be restricted without loss of generality to being of the form (8.4), we can also restrict our attention to subsumption problems of the form $C \sqsubseteq_{\mathcal{T}} \forall w.A$; instead of checking $C \sqsubseteq_{\mathcal{T}} D$ directly, it suffices to check whether $C \sqsubseteq_{\mathcal{T}} \forall w.A$ for every $\forall w.A \in \widehat{D}$.

The main idea underlying the PSpace subsumption algorithm presented below is the following: if $C \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$ then either $\forall w_0.A_0 \in \widehat{C}$, or there is some GCI

$$\forall v_1.A_1 \sqcap \cdots \sqcap \forall v_s.A_s \sqsubseteq \forall v.A_0$$

with $w_0 = pv$ for some $p \in \mathsf{N_R}^*$ and $C \sqsubseteq_\mathcal{T} \forall pv_i.A_i$ for $i = 1,\dots,s$. This idea is formalized using the notion of a derivation tree.

Definition 8.23. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox, C an $\mathcal{FL}_0$ concept description, and $\forall w_0.A_0$ a value restriction. A* derivation tree *for $\forall w_0.A_0$ w.r.t. $\mathcal{T}$ is a finite tree T satisfying the following properties:*

1. *The nodes of T are labeled with value restrictions, where the root is labeled with $\forall w_0.A_0$.*
2. *If $\forall w.A$ labels a node k of T and $\forall w_1.A_1,\dots,\forall w_s.A_s$ are the labels of its children $k_1,\dots,k_s$, then there is a GCI g in $\mathcal{T}$ of the form:*

$$g: \quad \forall v_1.A_1 \sqcap \dots \sqcap \forall v_s.A_s \sqsubseteq \forall v.A$$

and a word $p \in \mathsf{N_R}^*$ such that $w = pv$ and $w_i = pv_i$ for all $i = 1,\dots,s$. Each child node k_i with label $\forall w_i.A_i$ is assigned the following two additional labels: the GCI-used *$\mathfrak{g}(k_i) = g$ and the* prefix *$\mathfrak{d}(k_i) = p$. For the root k_0 we set $\mathfrak{g}(k_0) = \bot$ (standing for "no GCI") and $\mathfrak{d}(k_0) = w_0$.*

We denote as $\mathfrak{T}_\mathcal{T}(\forall w_0.A_0)$ the set of all derivation trees for $\forall w_0.A_0$ w.r.t. $\mathcal{T}$. The set of value restrictions labeling the leaves of such a tree T is denoted as $\mathfrak{L}(T)$. We say that T is a derivation tree for $C \sqsubseteq \forall w_0.A_0$ w.r.t. $\mathcal{T}$ *if $\mathfrak{L}(T) \subseteq \widehat{C}$, and denote the set of such trees with $\mathfrak{T}_{\mathcal{T},C}(\forall w_0.A_0)$. Finally, $\mathfrak{V}_{\mathcal{T},C}$ consists of the value restrictions $\forall w_0.A_0$ such that $\mathfrak{T}_{\mathcal{T},C}(\forall w_0.A_0) \neq \emptyset$.* ◇

Derivation trees can be used to obtain the following characterization of subsumption in $\mathcal{FL}_0$ w.r.t. TBoxes.

Lemma 8.24. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox, C an $\mathcal{FL}_0$ concept description, and $\forall w_0.A_0$ a value restriction. Then, $C \sqsubseteq_\mathcal{T} \forall w_0.A_0$ iff there exists a derivation tree for $C \sqsubseteq \forall w_0.A_0$ w.r.t. $\mathcal{T}$.*

Proof. The if-direction can be shown by a simple induction over the size of the derivation tree.

To prove the only-if direction, we use the set $\mathfrak{V}_{\mathcal{T},C}$ to construct a functional interpretation as follows: let $\mathcal{J}_{\mathcal{T},C}$ be the functional interpretation such that

$$A^{\mathcal{J}_{\mathcal{T},C}} := \{w \mid \forall w.A \in \mathfrak{V}_{\mathcal{T},C}\} \text{ for all } A \in \mathsf{N_C}.$$

We show that $\mathcal{J}_{\mathcal{T},C}$ is a model of $\mathcal{T}$. Let $E \sqsubseteq F \in \mathcal{T}$ and $w \in \Delta^{\mathcal{J}_{\mathcal{T},C}}$ be such that $w \in E^{\mathcal{I}_{\mathcal{T},C}}$. We can assume that $E \sqsubseteq F$ is of the form (8.4). Since $w \in E^{\mathcal{J}_{\mathcal{T},C}}$, this means that $wv_i \in (A_i)^{\mathcal{J}_{\mathcal{T},C}}$ for all $i, 1 \leq i \leq s$. Hence, by the definition of $\mathcal{J}_{\mathcal{T},C}$, we have that $\forall wv_i.A_i \in \mathfrak{V}_{\mathcal{T},C}$ for all $i, 1 \leq i \leq s$. By definition of $\mathfrak{V}_{\mathcal{T},C}$ we thus know that there exist derivation trees $T_1 \in \mathfrak{T}_{\mathcal{T},C}(\forall wv_1.A_1),\dots,T_s \in \mathfrak{T}_{\mathcal{T},C}(\forall wv_s.A_s)$. From this, it is clear that one can build a derivation tree $T \in \mathfrak{T}_{\mathcal{T},C}(\forall wv.A)$. This implies that $wv \in A^{\mathcal{I}_{\mathcal{T},C}}$ and thus $w \in (\forall v.A)^{\mathcal{J}_{\mathcal{T},C}}$. This shows that $\mathcal{J}_{\mathcal{T},C}$ is a model of $\mathcal{T}$.

Finally, notice that $\forall w.A \in \mathfrak{V}_{\mathcal{T},C}$ for all $\forall w.A \in \widehat{C}$. Hence, by the definition of $\mathcal{J}_{\mathcal{T},C}$, we have that $\varepsilon \in C^{\mathcal{J}_{\mathcal{T},C}}$. Since $C \sqsubseteq_\mathcal{T} \forall w_0.A_0$, this implies that $\varepsilon \in (\forall w_0.A_0)^{\mathcal{J}_{\mathcal{T},C}}$, and thus $w_0 \in A_0^{\mathcal{J}_{\mathcal{T},C}}$. This yields $\forall w_0.A_0 \in \mathfrak{V}_{\mathcal{T},C}$, which shows $\mathfrak{T}_{\mathcal{T},C}(\forall w_0.A_0) \neq \emptyset$ as required. □

Using this lemma, we can try to decide whether $C \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$ as follows. Start with the tree that has just one node k_0 labeled with $\forall w_0.A_0$. If this label belongs to $\widehat{C}$, then stop with success. Otherwise, try to find a GCI that allows to *expand the node* k_0 by adding children with appropriate labels (see 2. in Definition 8.23). If the subsumption $C \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$ holds, then there must be such a GCI. Thus, if there is none, we can stop with failure. Now assume that we have already generated a derivation tree T for $\forall w_0.A_0$. If $\mathfrak{L}(T) \subseteq \widehat{C}$, then we can stop with success. Otherwise, we pick a leaf whose label does not belong to $\widehat{C}$ and try to expand it using an appropriate GCI. If no such GCI exists, we stop with failure. Otherwise, we continue the expansion process until a failure case occurs or the labels of all leaves belong to $\widehat{C}$.

As described until now, this approach does not yield a PSPACE algorithm for subsumption for two reasons. First, the generated derivation trees may grow to having exponential size. Second, expansion need not terminate unless we install an appropriate cycle check, but then keeping the information necessary to detect cycles may require exponential space.

The main idea to solve the first problem is that we actually need not store the whole derivation tree T. It is sufficient to know the value restrictions in $\mathfrak{L}(T) \setminus \widehat{C}$ (together with the prefix label of the corresponding leaf). In fact, the value restrictions in $\mathfrak{L}(T) \setminus \widehat{C}$ are the ones that require further expansion. In order to ensure that this information can be represented using only polynomial space, we restrict the choice of which leaf is expanded next.

Definition 8.25. *Let T, T' be derivation trees for $\forall w_0.A_0$ w.r.t. $\mathcal{T}$. We write $T \to T'$ if T' is obtained from T by expanding a leaf whose label belongs to $\mathfrak{L}(T) \setminus \widehat{C}$, where among the eligible such leaves we choose one whose prefix label has maximal length.* ♢

The following proposition is an easy consequence of Lemma 8.24 and the fact that the order of leaf expansion is irrelevant.

Proposition 8.26. *Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox, C an $\mathcal{FL}_0$ concept description, and $\forall w_0.A_0$ a value restriction. Then, $C \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$ iff there exists a sequence $T_0 \to T_1 \to \ldots \to T_n$ of derivation trees for $\forall w_0.A_0$ w.r.t. $\mathcal{T}$ such that T_0 has just one node k_0 labeled with $\forall w_0.A_0$ and $T_n \in \mathfrak{T}_{\mathcal{T},C}(\forall w_0.A_0)$.* ♢

When checking for the existence of such a sequence $T_0 \to T_1 \to \ldots \to T_n$, it is sufficient to keep only $\mathfrak{L}(T_i) \setminus \widehat{C}$ as well as the prefix labels of the leaves that yield these value restrictions in memory.[3] Thus, our algorithm works with sets consisting of elements of the form $(\forall w.A, p)$ where $\forall w.A$ is the value restriction label and p is the prefix label of a leaf. We now show that the cardinality of these sets is polynomial in the size of $\mathcal{T}$, C, and $\forall w_0.A_0$, and that their elements come from an at most exponentially large base set. In fact, this then implies that there are only exponentially many such sets.

Given a derivation tree T, we denote the set of prefixes of nodes whose value restrictions belong to $\mathfrak{L}(T) \setminus \widehat{C}$ with $\mathcal{P}(T)$. In addition, we denote the prefix order on words with $\preceq$.

Lemma 8.27. *Let $\mathcal{T}$ be a forward TBox, and $T_0 \to T_1 \to \ldots \to T_n$ be a sequence of derivation trees for $\forall w_0.A_0$ w.r.t. $\mathcal{T}$ where T_0 is as described in Proposition 8.26. Then, for each $i, 0 \leq i \leq n$,*

1. *the elements of $\mathcal{P}(T_i)$ are linearly ordered by the prefix order $\preceq$;*

[3] Strictly speaking, different leaves could be labeled with the same value restriction $\forall w.A$, but clearly we need to expand only one of them into a derivation tree for $C \sqsubseteq \forall w.A$.

2. *the set $\mathfrak{L}(T_i) \setminus \widehat{C}$ contains at most $|w_0| \cdot t$ distinct value restrictions, where t is the number of distinct value restrictions occurring in the left-hand sides of the GCIs in $\mathcal{T}$.*

Proof. 1. Since T_0 has only one leaf, it trivially satisfies the property required by the lemma. Now assume that $(\mathcal{P}(T_i), \preceq)$ is a totally ordered set. Let ℓ be the leaf of T_i that is expanded when going from T_i to T_{i+1}, $\forall w.A$ its value restriction label and p its prefix label, $k_1, \ldots, k_s$ the children of ℓ in T_{i+1}, and p' the prefix label of these children. It is easy to see that $\mathcal{P}(T_{i+1}) \subseteq \mathcal{P}(T_i) \cup \{p'\}$. In addition, by the definition of node expansion, both p and p' are prefixes of w, and thus $p \preceq p'$ or $p' \prec p$. Since p is of maximal length in $\mathcal{P}(T_i)$ and $\mathcal{P}(T_i)$ is linearly ordered w.r.t. $\preceq$, all the elements of $\mathcal{P}(T_i)$ are prefixes of p. Thus, independently of whether $p \preceq p'$ or $p' \prec p$, the set $\mathcal{P}(T_i) \cup \{p'\}$ is also linearly ordered w.r.t. $\preceq$.

2. The restriction to forward TBoxes implies that each value restriction $\forall w.A$ occurring in a derivation tree of $\forall w_0.A_0$ satisfies $|w| \leq |w_0|$. Since each prefix is a prefix of such a word w, this length restriction also holds for the prefixes. A linearly ordered set of prefixes of length at most $|w_0|$ can clearly have at most $|w_0|$ elements. Finally, if $\forall w.A$ is the value restriction label of a leaf ℓ with prefix label u, then $w = uv_i$ where $\forall v_i.A_i$ occurs on the left-hand sides of some GCI in $\mathcal{T}$. □

The second part of this lemma shows that representing such a value restriction set $\mathfrak{L}(T_i) \setminus \widehat{C}$ requires only polynomial space. In addition, there can be only exponentially many different such sets. In fact, we have seen that the value restrictions $\forall w.A$ occurring in these sets satisfy $|w| \leq |w_0|$. In addition, these words w contain only role names occurring in the input. Thus, there are only exponentially many value restrictions that can potentially occur in these sets, and consequently there are only exponentially many possibilities for choosing a polynomial number of them. Our PSPACE algorithm thus non-deterministically generates a sequence $S_0, S_1, S_2, \ldots$ of such sets reflecting a sequence $T_0 \to T_1 \to T_2 \to \ldots$ of derivation trees, and keeps only the most recent such set in memory. To solve the termination issue, we do not test for cycles (since this would require keeping all the generated sets in memory), but in each step increment an appropriate exponential counter (which needs only polynomial space), which stops the algorithm with failure when it overflows. In fact, such an overflow indicates that there must be a repetition in the sequence (i.e., $i < j$ such that $S_i = S_j$), and thus a shorter sequence would yield the same result.

Below we formally present this NPSPACE algorithm for deciding subsumption w.r.t. forward $\mathcal{FL}_0$ TBoxes, and prove its correctness. Recall that t denotes the number of distinct value restrictions occurring in the left-hand sides of the GCIs in $\mathcal{T}$.

Lemma 8.28 (Termination). *Algorithm 2 is a terminating non-deterministic* PSPACE *procedure.*

Proof. Termination of the procedure is guaranteed due to the use of the counter c. To see that it is a non-deterministic PSPACE procedure, notice that each run of the procedure needs to store only the current content of the set S, the number k, and the counter c.

- Regarding the set S, it contains pairs of the form $(\forall w.A, p)$ where $p \preceq w$. According to Lemma 8.27, we have that the set $P = \{p \mid (\forall w.A, p) \in S\}$ is a linearly ordered set w.r.t. $\preceq$, and that $|w| \leq |w_0|$. Moreover $(\forall w.A, p) \in S$ only if $w = w_0$ and $A = A_0$, or $w = pv$ and $\forall v.A$ is a value restriction occurring in the left-hand side of a GCI in $\mathcal{T}$. Hence, S may contain at most $|w_0| \cdot t$ pairs, and the size of each pair is linear in $|w_0|$.

Algorithm 2: Subsumption in $\mathcal{FL}_0$ with respect to forward TBoxes

Input: Forward $\mathcal{FL}_0$ TBox $\mathcal{T}$, $\mathcal{FL}_0$ concept description C, and value restriction $\forall w_0.A_0$.
Output: "**yes**" if $C \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$, and "**fail**" otherwise.

1 **if** $\forall w_0.A_0 \in \widehat{C}$ **then**
2 | **return** ***yes***
3 **end**
4 $S := \{(\forall w_0.A_0, w_0)\}$;
5 $c := 0$ (c is stored in binary);
6 $k := ((|\mathsf{N_R}| + 1) \cdot t)^{|w_0|^2 \cdot t}$ (k is stored in binary);
7 **while** $S \neq \emptyset$ *and* $c \leq k$ **do**
8 | *non-deterministically* choose $(\forall w.A, p) \in S$ with longest p;
9 | *non-deterministically* choose a GCI $g : \forall v_1.A_1 \sqcap \cdots \sqcap \forall v_s.A_s \sqsubseteq \forall v.A$ such that $w = p'v$ for some $p' \in \mathsf{N_R}^*$;
10 | (**fail** if there is no such GCI);
11 | $S := (S \setminus \{(\forall w.A, p)\}) \cup \{(\forall p'v_i.A_i, p') \mid i = 1, \ldots, s \text{ and } \forall p'v_i.A_i \notin \widehat{C}\}$;
12 | $c := c + 1$;
13 **end**
14 **return** ***yes*** *if* $S = \emptyset$ *and* ***fail*** *otherwise*

- The number assigned to k is exponential in the size of $\mathcal{T}$, C and w_0, and the value of c never exceeds k. Hence, their binary representation can be stored in space polynomial in the size of $\mathcal{T}$, C and w_0.

Thus, every run of Algorithm 2 uses space polynomial in the size of $\mathcal{T}$, C and $\forall w_0.A_0$. □

The following two lemmata show that Algorithm 2 is sound and complete.

Lemma 8.29 (Soundness). *If Algorithm 2 answers* ***yes****, then* $C \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$ *holds.*

Proof. Assume that the algorithm has a successful run performing $n \geq 0$ iterations of the while loop. Let $S_0 = \{(\forall w_0.A_0, w_0)\}$ and $S_1, \ldots, S_n$ be the sets corresponding to S after the i^{th}-iteration of the while loop.

The following claim can easily be proved by *induction on* $n - i$:

$$\text{For all } 0 \leq i \leq n \text{ and all } (\forall w.A, p) \in S_i \text{ we have } C \sqsubseteq_{\mathcal{T}} \forall w.A.$$

Since the algorithm answers ***yes***, we have $S_n = \emptyset$, and thus the base case for $i = n$ is trivially true.

Assuming that it holds for $0 < i \leq n$, we prove that it also holds for $i - 1$. For this, let $(\forall w.A, p) \in S_{i-1}$. If $(\forall w.A, p) \in S_i$, the application of induction completes the proof. Otherwise, $(\forall w.A, p)$ is the pair selected by the algorithm to obtain S_i from S_{i-1}. This means that there exists a GCI $g : \forall v_1.A_1 \sqcap \ldots \sqcap \forall v_s.A_s \sqsubseteq \forall v.A$ in $\mathcal{T}$ such that $w = p'v$ for some $p' \in \mathsf{N_R}^*$. According to the algorithm, for each $\forall p'v_j.A_j$, either $(\forall p'v_j.A_j) \in \widehat{C}$ or $(\forall p'v_j.A_j, p') \in S_i$. In both cases, we have that $C \sqsubseteq_{\mathcal{T}} \forall p'v_j.A_j$ (the latter by induction). This means that $C \sqsubseteq_{\mathcal{T}} \forall p'v_1.A_1 \sqcap \ldots \sqcap \forall p'v_s.A_s$. Distributivity of $\forall p'$ over $\sqcap$ implies $C \sqsubseteq_{\mathcal{T}} \forall p'.(\forall v_1.A_1 \sqcap \ldots \sqcap \forall v_s.A_s)$. Hence, since $w = p'v$, the application of g yields $C \sqsubseteq_{\mathcal{T}} \forall w.A$.

Thus, since $S_0 = \{(\forall w_0.A_0, w_0)\}$, we obtain that $C \sqsubseteq_\mathcal{T} \forall w_0.A_0$. □

In principle, our proof of completeness considers a sequence $T_0 \to T_1 \to \ldots \to T_n$ of derivation trees for $\forall w_0.A_0$ w.r.t. $\mathcal{T}$ such that T_0 has just one node labeled with $\forall w_0.A_0$ and $T_n \in \mathfrak{T}_{\mathcal{T},C}(\forall w_0.A_0)$. It then transforms this sequence into a sequence of sets $S_0, S_1, \ldots, S_n$ by considering the value restrictions not in $\widehat{C}$ labelling the leaves of the trees T_i (together with the prefix label of the respective leaf). Unfortunately, this sequence of sets need not always be a sequence of sets that can be produced by our algorithm. In fact, it may be the case that T_i contains several leaves with label $\forall w.A$ and maximal prefix u. If one of these leaves is expanded in the transition $T_i \to T_{i+1}$, then the others remain in T_{i+1}, and thus their label $\forall w.A$ and prefix u remains in S_{i+1}. However, one can assume without loss of generality that all such leaves are expanded simultaneously in the same way. Let us denote the transition relation on derivation trees obtained this way by $\to_s$. It is easy to see that Proposition 8.26 and Lemma 8.27 also hold with $\to_s$ in place of $\to$.

Lemma 8.30 (Completeness). *If $C \sqsubseteq_\mathcal{T} \forall w_0.A_0$, then Algorithm 2 answers **yes**.*

Proof. Assume that $C \sqsubseteq_\mathcal{T} \forall w_0.A_0$. If $\forall w_0.A_0 \in \widehat{C}$, then Algorithm 2 immediately answers **yes**. Otherwise, by (the $\to_s$-variant of) Proposition 8.26, there exists a sequence $T_0 \to_s T_1 \to_s \ldots \to_s T_n$ of derivation trees for $\forall w_0.A_0$ w.r.t. $\mathcal{T}$ such that T_0 has just one node labeled with $\forall w_0.A_0$ and $T_n \in \mathfrak{T}_{\mathcal{T},C}(\forall w_0.A_0)$. We will now show how $T_0 \to_s T_1 \to_s \ldots \to_s T_n$ can be used to produce a successful run of the algorithm. Let us start by constructing a sequence of sets $S_0, S_1, \ldots, S_n$ as follows:

- $S_0 = \{(\forall w_0.A_0, w_0)\}$.
- For all $0 \leq i < n$, the set S_{i+1} is constructed from S_i as follows. Since $T_i \to_s T_{i+1}$, T_{i+1} is obtained from T_i by expanding one leaf ℓ_i of T_i with label $\forall w^i.A^i$ (or several such leaves) such that:
  - $\forall w^i.A^i \in \mathfrak{L}(T_i) \setminus \widehat{C}$ and $\mathfrak{d}(\ell_i) = p^i$ is of maximal length.
  - There exists a GCI g_i of the form $\forall v_1.A_1 \sqcap \ldots \sqcap \forall v_s.A_s \sqsubseteq \forall v.A^i$ such that $p'v = w^i$ for some $p' \in \mathsf{N_R}^*$ and $\forall p'v_j.A_j \in \mathfrak{L}(T_{i+1})$, $(1 \leq j \leq s)$.

  Then $S_{i+1} := (S_i \setminus \{(\forall w^i.A^i, p^i)\}) \cup \{(\forall p'v_i.A_i, p') \mid i = 1, \ldots, s \text{ and } \forall p'v_i.A_i \notin \widehat{C}\}$.

Using induction on i, we can show the following for all $i, 0 \leq i \leq n$:

$(\forall w.A, p) \in S_i$ only if there is a leaf ℓ in T_i with label $\forall w.A \in \mathfrak{L}(T_i) \setminus \widehat{C}$ and $\mathfrak{d}(\ell) = p$.

Consequently, $\mathfrak{L}(T_n) \subseteq \widehat{C}$ implies $S_n = \emptyset$. Furthermore, notice that starting with $S = S_i$, an iteration of the **while** loop resulting in $S = S_{i+1}$ can be achieved by choosing $(\forall w^i.A^i, p^i)$ from S and g_i from $\mathcal{T}$. Hence, if $n \leq k$ the sequence $S_0, \ldots, S_n$ yields a successful run of the algorithm on input $\mathcal{T}$, C and $\forall w_0.A_0$.

In case $n > k$, the counter c would overflow leading to a failing run of the algorithm. However, $S_0, \ldots, S_n$ can be transformed into a sufficiently short sequence $S_0, S_{j_1}, \ldots, S_{j_m}$ inducing a successful run. This is an easy consequence of the following two arguments:

- Suppose there are two indices $0 \leq i_1 < i_2 \leq n$ such that $S_{i_1} = S_{i_2}$. Then, we can transform $S_0, \ldots, S_n$ into a shorter sequence $S_0, \ldots, S_{i_1}, S_{i_2+1}, \ldots, S_n$ satisfying the same

properties described above for $S_0, \ldots, S_n$. Iteratively applying this argument will result in a sequence $S_0, S_{j_1}, \ldots, S_{j_m}$, where $1 \leq j_i \leq n$ for all $1 \leq i \leq m$ and $S_{j_{i_1}} \neq S_{j_{i_2}}$ for all $0 \leq i_1 < i_2 \leq m$.

- As seen in Lemma 8.28, each set S_i contains at most $|w_0| \cdot t$ elements. In addition, there are at most $(|\mathsf{N_R}| + 1)^{|w_0|} \cdot t$ many different elements a set S_i can contain, where we use $(|\mathsf{N_R}| + 1)^{|w_0|}$ as an over-approximation of the number of words of length at most $|w_0|$ over the alphabet $\mathsf{N_R}$. This means that there are at most $((|\mathsf{N_R}| + 1)^{|w_0|} \cdot t)^{|w_0| \cdot t} \leq ((|\mathsf{N_R}| + 1) \cdot t)^{|w_0|^2 \cdot t}$ different such sets.

Hence, $m \leq k$ and thus $S_0, S_{j_1}, \ldots, S_{j_m}$ corresponds to a successful run of Algorithm 2 on input $\mathcal{T}$, C and $\forall w_0.A_0$. □

Using the fact that PSPACE = NPSPACE, we thus have shown the desired PSPACE upper bound for subsumption.

Basically the same algorithm can also be used to handle *backward TBoxes*, which consist of GCIs of the form (8.4) where $|v_i| \geq |v|$ for all $i, 1 \leq i \leq s$. While for such TBoxes expansion of leaves may increase the length of value restrictions, one can stop with failure whenever a value restriction is generated that is longer than the longest value restriction in $\widehat{C}$.

Theorem 8.31. *Subsumption in $\mathcal{FL}_0$ w.r.t. forward (backward) TBoxes is in* PSPACE.

Matching in $\mathcal{FL}_0$ w.r.t. forward TBoxes

We now adapt the NPSPACE subsumption algorithm for forward TBoxes introduced above to the problem of deciding whether a matching problem has an $\mathcal{FL}_0$ matcher. Consider a matching problem of the form (8.2). According to Lemma 8.18, we need to check the two conditions stated in this lemma. The first condition is a subsumption test w.r.t. the forward TBox $\mathcal{T}$, and can thus be performed in PSPACE. Regarding the second test, we add elements to the finite language $L_{i,j}$ on the fly while performing the subsumption test $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} C$.

Basically, we run the NPSPACE subsumption algorithm on $E \sqsubseteq \forall w_0.A_0$ for every value restriction $\forall w_0.A_0$ in $\widehat{C}$. But we now have two conditions under which a leaf with label $\forall w.A_j$ need not be expanded: either $\forall w.A_j \in \widehat{E}$, or there exists $i, 1 \leq i \leq m$, $w_1 \in L_i$, and $w_2 \in \widehat{L}_{i,j} = \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$ with $w = w_1 w_2$. Checking the second condition requires only polynomial space. In fact, there are only polynomially many pairs w_1, w_2 to be considered. For each of them, we need to check for all $u \in L_i$ whether $uw_2 \in \mathcal{L}_{\mathcal{T}}(C, A_j)$. Each of these tests is a subsumption test $C \sqsubseteq_{\mathcal{T}} \forall uw_2.A_j$, which needs only polynomial space. Note that for backward TBoxes $\mathcal{T}$, the languages $\mathcal{L}_{\mathcal{T}}(C, A_j)$ are actually finite and contain only words not longer than the longest value restriction in $\widehat{C}$.

Algorithm 3 below formally describes the decision procedure sketched above to solve the matching problem w.r.t. forward TBoxes.

Lemma 8.32 (Termination). *Algorithm 3 is a terminating non-deterministic* PSPACE *procedure.*

Proof. Termination of the algorithm is guaranteed because of the counter and the fact that there are finitely many value restrictions in $\widehat{C}$. To see that it is a non-deterministic PSPACE

Algorithm 3: Matching in $\mathcal{FL}_0$ with respect to forward TBoxes

Input: A forward $\mathcal{FL}_0$ TBox $\mathcal{T}$ and a matching problem of the form (8.2).
Output: **"yes"** if $C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_m.X_m$ has an $\mathcal{FL}_0$ matcher, and **"fail"** otherwise.

1 **if** $C \not\sqsubseteq_{\mathcal{T}} E$ **then**
2 **fail**
3 **end**
4 **foreach** $\forall w_0.A_0 \in \widehat{C} \setminus \widehat{E}$ **do**
5 $S := \{(\forall w_0.A_0, w_0)\}$;
6 $c := 0$ (c is stored in binary);
7 $k := ((|\mathsf{N_R}| + 1) \cdot t)^{|w_0|^2 \cdot t}$ (k is stored in binary);
8 **while** $S \neq \emptyset$ *and* $c \leq k$ **do**
9 *non-deterministically* choose $(\forall w.A_j, p) \in S$ with longest p;
10 **if** *there exists* $i, 1 \leq i \leq m$, $w_1 \in L_i$, $w_2 \in \widehat{L}_{i,j} = \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$ *with* $w = w_1 w_2$ **then**
11 $S := S \setminus \{(\forall w.A_j, p)\}$
12 **else**
13 *non-deterministically* choose a GCI $g : \forall v_1.A_1 \sqcap \cdots \sqcap \forall v_s.A_s \sqsubseteq \forall v.A$ such that $w = p'v$ for some $p' \in \mathsf{N_R}^*$;
14 (**fail** if there is no such GCI);
15 $S := (S \setminus \{(\forall w.A, p)\}) \cup \{(\forall p' v_i.A_i, p') \mid i = 1, \ldots, s \text{ and } \forall p' v_i.A_i \notin \widehat{E}\}$;
16 **end**
17 **end**
18 **fail** if $S \neq \emptyset$
19 **end**
20 **return** *yes*

procedure, notice that, like Algorithm 2, it only needs to store the content of the set S, the counter c and the number k, and be able to perform the check in line 10 in polynomial space.

The proof that we only need polynomial space to store S is the same as in Lemma 8.28. For the check in line 10, note that there are only polynomially many pairs w_1, w_2 to be considered. For each of them, we need to check for all $u \in L_i$ whether $uw_2 \in \mathcal{L}_{\mathcal{T}}(C, A_j)$. Each of these tests is a subsumption test $C \sqsubseteq_{\mathcal{T}} \forall uw_2.A_j$, which needs only polynomial space according to Theorem 8.31. Hence, the entire algorithm needs only polynomial space. □

Lemma 8.33 (Soundness). *If Algorithm 3 answers* ***yes****, then* $C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_m.X_m$ *has an* $\mathcal{FL}_0$ *matcher.*

Proof. For all $1 \leq i \leq m$ and $1 \leq j \leq k$ we define the set $L_{i,j} \subseteq \mathsf{N_R}^*$ as follows:

$$L_{i,j} := \{w_2 \mid \text{the check in line 10 was invoked for } w_2 \text{ and succeeded}\}.$$

Clearly, $L_{i,j} \subseteq \widehat{L}_{i,j}$ and the sets $L_{i,j}$ are finite since line 10 is reached only finitely often during a successful run of the procedure. Using these sets, we define the following substitution:

$$\sigma(X_i) = \prod_{j=1}^{k} \forall L_{i,j}.A_j, 1 \leq i \leq m.$$

We use Lemma 8.18 to show that σ is a solution of the matching problem. Since the algorithm did not fail, we know that $C \sqsubseteq_{\mathcal{T}} E$. Hence, it remains to show the second condition that $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} C$, i.e., $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$ for each $\forall w_0.A_0 \in \widehat{C}$.

If $\forall w_0.A_0 \in \widehat{E}$, then this subsumption holds trivially. If $\forall w_0.A_0 \in \widehat{C} \setminus \widehat{E}$, then let $S_0, \ldots, S_n$ be the sequence of sets S computed by the corresponding iteration of the while loop. Similarly to the proof of Lemma 8.29, induction can be used to show that $(\forall w.A, p) \in S_i$ implies $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} \forall w.A$. Since $S_0 = \{(\forall w_0.A_0, w_0)\}$, this implies $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$. Thus, we can conclude that $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} C$. □

Lemma 8.34 (Completeness). *If $C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_m.X_m$ has an $\mathcal{FL}_0$ matcher, then Algorithm 3 answers **yes**.*

Proof. By Lemma 8.18, the following two conditions are satisfied:

1. $C \sqsubseteq_{\mathcal{T}} E$.
2. There are finite languages $L_{i,j} \subseteq \widehat{L}_{i,j}$ such that $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} C$, where σ is defined as in (8.3).

The first condition guarantees that the test in Line 1 does not fail. From the second one, we know that $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m) \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$ for all $\forall w_0.A_0 \in \widehat{C}$. By applying distributivity of value restrictions over conjunction, $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_m.\sigma(X_m)$ can be transformed into an equivalent concept description D that is a conjunction of value restrictions. Note that

$$\forall w.A_j \in \widehat{D} \quad \text{iff} \quad \begin{array}{l} \text{either } \forall w.A_j \in \widehat{E}, \text{ or there exist } 1 \leq i \leq m \text{ and } w_1, w_2 \in \mathsf{N_R}^* \\ \text{such that } w = w_1 w_2,\ w_1 \in L_i \text{ and } w_2 \in L_{i,j} \subseteq \widehat{L}_{i,j}. \end{array} \tag{8.5}$$

By Lemma 8.30, $D \sqsubseteq_{\mathcal{T}} \forall w_0.A_0$ implies that there is a successful run of Algorithm 2 for the corresponding subsumption query. Due to (8.5), this is also a successful run of Algorithm 3 for the matching problem $C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_m.X_m$. □

Again, backward TBoxes can be treated similarly. Finally, we obtain the main result of this section.

Theorem 8.35. *Matching in $\mathcal{FL}_0$ w.r.t. forward (backward) TBoxes is in* PSPACE.

8.5 Outlook

We have shown in this chapter that matching in $\mathcal{FL}_0$ w.r.t. TBoxes is in EXPTIME, thus complementing the positive results for matching w.r.t. TBoxes in $\mathcal{EL}$ [BM14]. This is the best

possible complexity for matching in this setting since already the subsumption problem is ExpTime-hard. One drawback of our approach is the fact that the use of compactness in Lemma 8.19 does not provide us with a constructive algorithm to compute $\mathcal{FL}_0$ matchers. In particular, we do not even obtain a bound on the size of such matchers, which could potentially provide us with such a procedure. Furthermore, we have shown that the complexity of subsumption and matching can be lowered to PSpace if restricted kinds of TBoxes are considered. Unfortunately, until now we could not show a matching PSpace lower bound, but we believe that for forward TBoxes these problems are indeed PSpace-complete.

The big open problem in this area is unification in $\mathcal{FL}_0$ w.r.t. TBoxes, for which nothing is known. Note that our approach fails in this setting: the unification problem (w.r.t. the empty TBox)

$$X \equiv^? A \sqcap \forall r.X$$

has the $\mathcal{FL}_{\text{reg}}$ substitution $\sigma = \{X \mapsto \forall\{r\}^*.A\}$ is a unifier, but no $\mathcal{FL}_0$ unifier. For $\mathcal{EL}$, decidability of unification w.r.t. TBoxes is also an open problem, but there are positive results for TBoxes satisfying certain restrictions on cyclic dependencies [BBM12]. It would be interesting to see whether this restriction or the restrictions we imposed in Section 8.4 of the present chapter can lead to positive results for unification in $\mathcal{FL}_0$ w.r.t. TBoxes.

Chapter 9

Conclusion

As a conclusion to this thesis, we summarize the results obtained, while also discussing extensions of the current work for future consideration.

9.1 Contributions of the thesis and future work

We started our investigation in the simpler setting of the equational theory ACUI in Chapter 2. We extended ACUI-unification in two orthogonal directions, which we then brought together. We first introduced approximate ACUI-unification and explored the computational complexity of the problem w.r.t. three different measures. For two of these measures, the complexity increases from P to NP-complete, whereas for one of them it stays in P. We moved on to study unification in equational theories that are obtained from ACUI by adding a finite set G of ground identities. We were able to show that ACUIG-unification is of the same complexity as for ACUI, i.e., polynomial. Finally, we investigated approximate ACUIG-unification, combining the two extensions. For the measures for which already approximate ACUI-unification is NP-complete, the same holds for approximate ACUIG-unification, while for the third measure, the exact complexity depends on the particular set G of ground identities. We identified a theory G for which the problem is NP-complete, but also a class of theories G for which approximate ACUIG-unification is in P. Figure 9.1 contains a schematic description of our results extending ACUI-unification.

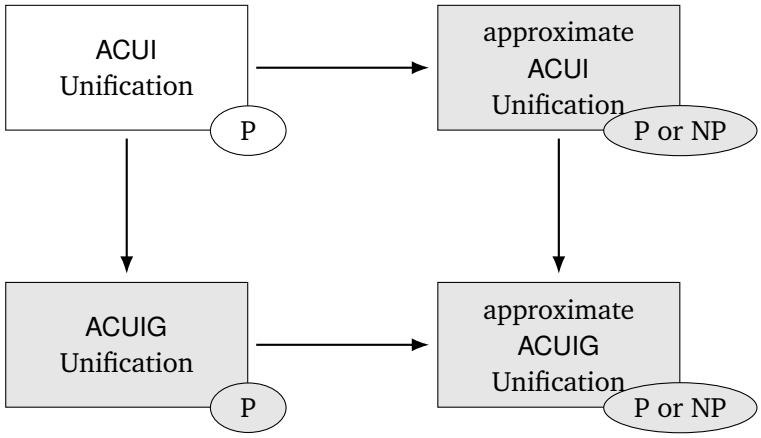

Figure 9.1: Extensions of ACUI unification. The results obtained in this thesis are highlighted.

The main purpose of Chapter 3 was to provide the necessary background from formal language and automata theory that would be needed for the rest of the thesis. The novel contributions were quite technical. We initially described how infinite trees can be used to represent tuples of languages on a theoretical level. We then studied how looping tree automata can be used to (uniquely) define such trees. Next, we provided a direct link from tuples of languages to LTAs by constructing an LTA accepting (the unique tree corresponding to) a tuple of languages from the DFAs accepting these languages.

Furthermore, we demonstrated how weighted tree automata with discounting can be used to express language distances. For this purpose, we used weighted tree automata with discounting, and reduced the problem of computing the distance to the problem of computing the behavior of such an automaton on the unlabeled infinite tree. If the weights of the automaton come from the semiring $\mathbb{R}_{\text{inf}}$, then this behavior can be computed in polynomial time provided that the employed discounting is nondecreasing or contracting. An obvious topic for future research is thus to extend these results to discounting that is neither contracting nor nondecreasing, or to other semirings as weight structures.

The main technical contribution of Chapter 5 is the development of a general framework for defining concept distance measures (CDMs) for the DL $\mathcal{FL}_0$ that are computable and invariant under equivalence w.r.t. general TBoxes. Our framework is based on a characterization of equivalence w.r.t. general $\mathcal{FL}_0$ TBoxes that uses tuples of formal languages. In particular, we showed how a language distance can be paired with an appropriate combining function to yield a proper CDM. In addition, we proved that our framework guarantees that the obtained CDMs satisfy further properties often required for concept similarity measures.

Furthermore, making use of the results from Chapter 3 we proved computability of such CDMs in polynomial time in case the corresponding language distance is expressed by a wLTA.

We have used this framework in Chapter 6, but only in the very limited setting of the empty TBox. It seems plausible that our framework can be used to define CSMs as well, if instantiated with wLTAs inducing suitable functions. It would be interesting to see whether it can output measures that can be employed within one of the other approximation approaches mentioned in Section 1.3.

In Chapter 6 we have extended unification in $\mathcal{FL}_0$ to approximate unification (w.r.t. the empty TBox). The degree of approximation was defined using a CDM. We demonstrated that for CDMs defined using language distances, i.e., obtainable by our framework, the problem can be reduced to approximate solving language equations if the underlying language distance satisfies certain properties. For a class of such distances we showed decidability of the latter problem in time exponential in the size of the instance. For the distances d_1 and d_2 we showed in particular that the problem actually is complete for ExpTime. Finally, we laid out some ideas on how our approach can be used to obtain approximate solutions and how to solve a variation of the decision problem.

Interesting topics for future research are considering approximate unification for other DLs for which unification has been investigated. Such instances are the logics $\mathcal{FL}_{\text{reg}}$ and $\mathcal{FL}\bot_{reg}$ that extend $\mathcal{FL}_0$ and for which unification is ExpTime-complete ([BK01] and [BK02] respectively), as well as the dual logic $\mathcal{EL}$, for which unification is NP-complete [BM10]. For this to become possible, however, one would need to define appropriate concept distance measures in these logics. It would be interesting to see whether one can use similarity measures for this purpose.

On the formal language level, we could explore how our results can be used and extended to investigate approximately solving other kinds of language equations [Kun07].

Furthermore, the study of approximate unification in $\mathcal{FL}_0$ is, in a sense, orthogonal to the one about ACUI. While the first only focuses on how to approximately solve single equations, the latter deals with how to deal with systems of equations.

In the classical case, this is without loss of generality for $\mathcal{FL}_0$, since one can reduce a system of several equations to be unified to a single one. In fact, not only solvability of the single equation is equivalent to solvability of the system, but also any substitution that is a solution in the one case is also a solution for the other.[1] In the approximate setting, however, one has multiple ways to combine the violations from each of the several equations into a single *degree of violation* for the entire system to compare to the given threshold. In principle, it should be possible to implement the ideas from Chapter 2 in the setting of approximate $\mathcal{FL}_0$ unification.

In Chapter 7 we showed that the problem of approximate matching in $\mathcal{FL}_0$ (w.r.t. the empty TBox) is contained in NP for a wide class of distances. Furthermore, we obtained sharp bounds for certain language distances, proving the problem to be solvable in polynomial time for d_1, while it is NP-complete for d_0 and d_2. In order to achieve the latter hardness results, we devised an NP-complete satisfiability problem, which, to the best of our knowledge is novel.

Matching in extensions of $\mathcal{FL}_0$ has been investigated in [BKB+99], where the problem reduces to solving equations that involve infinite languages. Hence, it is worth investigating whether the results of this chapter can be extended to such equations.

In Chapter 8 we showed that matching in $\mathcal{FL}_0$ w.r.t. TBoxes is in ExpTime, thus complementing the positive results for matching w.r.t. TBoxes in $\mathcal{EL}$ [BM14]. This is the best possible complexity for matching in this setting since already the subsumption problem is ExpTime-hard. One drawback of our approach is the fact that the use of compactness in Lemma 8.19 does not provide us with a constructive algorithm to compute $\mathcal{FL}_0$ matchers. In particular, we do not even obtain a bound on the size of such matchers, which could potentially provide us with such a procedure. Furthermore, we have shown that the complexity of subsumption and matching can be lowered to PSpace if restricted kinds of TBoxes are considered. Unfortunately, until now we could not show a matching PSpace lower bound, but we believe that for forward TBoxes these problems are indeed PSpace-complete.

The big open problem in this area is unification in $\mathcal{FL}_0$ w.r.t. TBoxes, for which nothing is known. For $\mathcal{EL}$, decidability of unification w.r.t. TBoxes is also an open problem, but there are positive results for TBoxes satisfying certain restrictions on cyclic dependencies [BBM12]. It would be interesting to see whether this restriction or the restrictions we imposed in Section 8.4 can lead to positive results for unification in $\mathcal{FL}_0$ w.r.t. TBoxes.

Another natural continuation of this research would be to try and extend these results on matching to other DLs. An evident choice would be $\mathcal{FL}_{\text{reg}}$, since it already makes an appearance in the proofs for $\mathcal{FL}_0$. Note, however, that we only consider $\mathcal{FL}_{\text{reg}}$ concept descriptions in the presence of an $\mathcal{FL}_0$ TBox. Hence our techniques might not suffice for tackling this problem. Other candidate logics would be the extentions of $\mathcal{FL}_0$ from [BKB+99].

Finally, it would definitely make sense to combine the two extensions, as we did for ACUI, and investigate approximate $\mathcal{FL}_0$ unification (or matching) in the presence of TBoxes. Note

[1] See [BN01] for details.

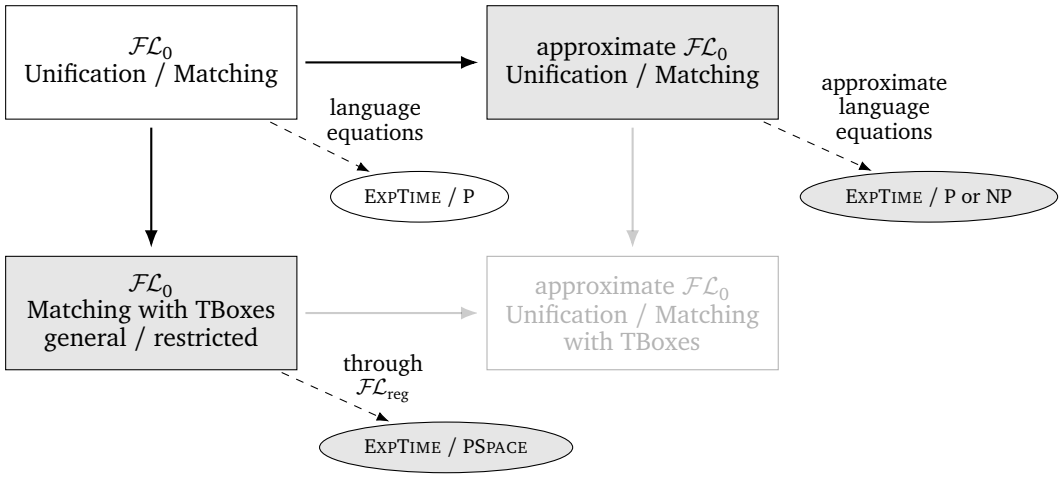

Figure 9.2: Extensions of $\mathcal{FL}_0$ unification and matching. The results obtained in this thesis are highlighted. In particular, approximate unification has been investigated in Chapter 6, approximate matching in Chapter 7, and matching in the presence of general and restricted TBoxes in Chapter 8.

that the CDMs we have defined in Chapter 5 can deal with general TBoxes, and hence can be used to define this problem. Other than this, however, we do not have a clear idea on how to tackle this problem. One exception is approximate unification w.r.t. (CDMs using) the language distance d_1 and forward TBoxes. In particular, the definition of d_1 guarantees that we only need to check for violations up to a certain role depth (that is linearly defined by the input threshold), and forward TBoxes ensure that longer chains of value restrictions cannot have an influence on this. Hence, we can restrict our search for unifiers to substitutions up to this depth. Since such substitutions can be of exponential size, and checking subsumption (and thus also equivalence) w.r.t. a forward TBox can be performed in space polynomial in the size of the input concept descriptions, we obtain a (potentially very naïve) 2ExpSpace algorithm. This could be a first step towards a more detailed investigation.

An overview of the results extending $\mathcal{FL}_0$ unification and matching can be seen in Figure 9.2.

Bibliography

[AHM03] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar: 'Discounting the Future in Systems Theory'. In *Proc. of the 30th Int. Coll. on Automata, Languages and Programming (ICALP 2003)*. Volume 2719. Lecture Notes in Computer Science. Springer, 2003, pages 1022–1037 (cited on page 56).

[BKB+99] F. Baader, R. Küsters, A. Borgida, and D. McGuinness: 'Matching in Description Logics'. In *Journal of Logic and Computation* **9**(3): 1999, pages 411–447 (cited on pages 120, 143).

[Baa96] Franz Baader: 'Using Automata Theory for Characterizing the Semantics of Terminological Cycles'. In *Annals of Mathematics and Artificial Intelligence* **18**: 1996, pages 175–219 (cited on pages 2, 3, 7, 11).

[BBM12] Franz Baader, Stefan Borgwardt, and Barbara Morawska: 'Extending Unification in $\mathcal{EL}$ towards General TBoxes'. In *Proc. of the 13th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2012)*. AAAI Press/The MIT Press, 2012, pages 568–572 (cited on pages 12, 140, 143).

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz: 'Pushing the $\mathcal{EL}$ Envelope'. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*. Edited by Leslie Pack Kaelbling and Alessandro Saffiotti. Edinburgh (UK): Morgan Kaufmann, Los Altos, 2005, pages 364–369 (cited on pages 77, 86, 121, 122, 127).

[BBF15] Franz Baader, Gerhard Brewka, and Oliver Fernández Gil: 'Adding Threshold Concepts to the Description Logic $\mathcal{EL}$'. In *Proc. of the 10th Int. Symp. on Frontiers of Combining Systems (FroCoS 2015)*. Volume 9322. Lecture Notes in Computer Science. Springer, 2015, pages 33–48 (cited on page 6).

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors: *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press, 2003 (cited on pages 1, 75).

[BE16] Franz Baader and Andreas Ecke: 'Reasoning with Prototypes in the Description Logic ALC using Weighted Tree Automata'. In *Proceedings of the 10th International Conference on Language and Automata Theory and Applications (LATA 2016)*, Prague, Czech Republic. Volume 9618. Lecture Notes in Computer Science. Springer-Verlag, 2016, pages 63–75 (cited on page 6).

[BFM17] Franz Baader, Oliver Fernández Gil, and Pavlos Marantidis: 'Approximation in Description Logics: How Weighted Tree Automata Can Help to Define the Required Concept Comparison Measures in $\mathcal{FL}_0$'. In *Proceedings of the 11th International Conference on Language and Automata Theory and Applications (LATA 2017)*, Umeå, Sweden. Edited by Frank Drewes, Carlos Martín-Vide, and Bianca Truthe. Volume 10168. Lecture Notes in Computer Science. Springer, 2017, pages 3–26 (cited on page 8).

[BFM18] Franz Baader, Oliver Fernández Gil, and Pavlos Marantidis: 'Matching in the Description Logic $\mathcal{FL}_0$ with respect to General TBoxes'. In *Proc. of the 22nd Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'18)*. Edited by Gilles Barthe, Geoff Sutcliffe, and Margus Veanes. Volume 57. EPiC Series in Computing. EasyChair, 2018, pages 76–94 (cited on page 9).

[BFP18] Franz Baader, Oliver Fernández Gil, and Maximilian Pensel: 'Standard and Non-Standard Inferences in the Description Logic $\mathcal{FL}_0$ Using Tree Automata'. In *GCAI 2018, 4th Global Conference on Artificial Intelligence*. Edited by Daniel Lee, Alexander Steen, and Toby Walsh. Volume 55. EPiC Series in Computing. EasyChair, 2018, pages 1–14 (cited on pages 2, 7, 77, 78, 124–127).

[BK01] Franz Baader and Ralf Küsters: 'Unification in a Description Logic with Transitive Closure of Roles'. In *Logic for Programming, Artificial Intelligence, and Reasoning*. Edited by Robert Nieuwenhuis and Andrei Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pages 217–232 (cited on page 142).

[BK02] Franz Baader and Ralf Küsters: 'Unification in a Description Logic with Inconsistency and Transitive Closure of Roles'. In *Proceedings of the 2002 International Workshop on Description Logics (DL2002), Toulouse, France, April 19-21, 2002*. 2002 (cited on page 142).

[BKM99] Franz Baader, Ralf Küsters, and Ralf Molitor: 'Computing Least Common Subsumers in Description Logics with Existential Restrictions'. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*. 1999, pages 96–101 (cited on page 11).

[BKM00] Franz Baader, Ralf Küsters, and Ralf Molitor: 'Rewriting Concepts Using Terminologies'. In *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000*. 2000, pages 297–308 (cited on page 6).

[BM17] Franz Baader and Pavlos Marantidis: 'Language equations for approximate matching in the Description Logic $\mathcal{FL}_0$'. In *Proceedings of the 31st International Workshop on Unification (UNIF'17)*. Edited by Adrià Gascón and Christopher Lynch. Oxford, UK, 2017 (cited on page 9).

[BMM18] Franz Baader, Pavlos Marantidis, and Antoine Mottet: 'ACUI Unification modulo Ground Theories'. In *Proceedings of the 32th International Workshop on Unification (UNIF 2018)*. Edited by Mauricio Ayala-Rincón and Philippe Balbiani. Oxford, UK, 2018, pages 37–41 (cited on page 7).

[BMO16a] Franz Baader, Pavlos Marantidis, and Alexander Okhotin: 'Approximate Unification in the Description Logic $\mathcal{FL}_0$'. In *Proc. of the 15th Eur. Conf. on Logics in Artificial Intelligence (JELIA 2016)*. Edited by Loizos Michael and Antonis C. Kakas. Volume 10021. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2016, pages 49–63 (cited on page 9).

[BMO16b] Franz Baader, Pavlos Marantidis, and Alexander Okhotin: 'Approximately Solving Set Equations'. In *Proceedings of the 30th International Workshop on Unification (UNIF'16)*. Edited by Silvio Ghilardi and Manfred Schmidt-Schauß. Porto, Portugal, 2016, pages 37–41 (cited on page 7).

[BM09] Franz Baader and Barbara Morawska: 'Unification in the Description Logic $\mathcal{EL}$'. In *Proc. of the 20th Int. Conf. on Rewriting Techniques and Applications (RTA 2009)*. Edited by Ralf Treinen. Volume 5595. Lecture Notes in Computer Science. Springer, 2009, pages 350–364 (cited on pages 11, 12).

[BM10] Franz Baader and Barbara Morawska: 'Unification in the Description Logic $\mathcal{EL}$'. In *Logical Methods in Computer Science* **6**(3): 2010 (cited on pages 5, 142).

[BM14] Franz Baader and Barbara Morawska: 'Matching with Respect to General Concept Inclusions in the Description Logic *EL*'. In *KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings*. 2014, pages 135–146 (cited on pages 12, 139, 143).

[BN01] Franz Baader and Paliath Narendran: 'Unification of Concept Terms in Description Logics'. In *J. of Symbolic Computation* **31**(3): 2001, pages 277–305 (cited on pages 2, 5, 7–9, 11, 12, 50, 77, 78, 80–82, 93, 103, 111, 112, 115, 116, 124, 127, 143).

[BN98] Franz Baader and Tobias Nipkow: *Term Rewriting and All That*. United Kingdom: Cambridge University Press, 1998 (cited on page 13).

[BN96] Franz Baader and Werner Nutt: 'Combination Problems for Commutative/Monoidal Theories or How Algebra Can Help in Equational Unification'. In *Applicable Algebra in Engineering, Communication and Computing* **7**: Mar. 1996, pages 309–337 (cited on page 3).

[BO12] Franz Baader and Alexander Okhotin: 'Solving Language Equations and Disequations with Applications to Disunification in Description Logics and Monadic Set Constraints'. In *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*. 2012, pages 107–121 (cited on page 73).

[BO13] Franz Baader and Alexander Okhotin: 'On Language Equations with One-sided Concatenation'. In *Fundamenta Informaticae* **126**(1): 2013, pages 1–35 (cited on pages 7, 8, 50, 53, 54, 82–84, 94, 97, 98, 103, 104).

[BP10] Franz Baader and Rafael Peñaloza: 'Automata-based Axiom Pinpointing'. In *J. of Automated Reasoning* **45**(2): 2010, pages 91–129 (cited on pages 64, 74).

[BS93] Franz Baader and Klaus U. Schulz: 'General A- and AX-unification via optimized combination procedures'. In *Word Equations and Related Topics*. Edited by Habib Abdulrab and Jean-Pierre Pécuchet. Volume 677. Lecture Notes in Computer Science. Springer, 1993, pages 23–42 (cited on page 24).

[BS96] Franz Baader and Klaus U. Schulz: 'Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures'. In *J. Symbolic Computation* **21**: 1996, pages 211–243 (cited on pages 12, 31).

[BS94] Franz Baader and Jörg H. Siekmann: 'Unification Theory'. In *Handbook of Logic in Artificial Intelligence and Logic Programming*. Edited by D. M. Gabbay, C. J. Hogger, and J. A. Robinson. Oxford, UK: Oxford University Press, 1994, pages 41–125 (cited on pages 3, 4, 11).

[BS01] Franz Baader and Wayne Snyder: 'Unification Theory'. In *Handbook of Automated Reasoning*. Edited by J.A. Robinson and A. Voronkov. Volume I. Elsevier Science Publishers, 2001, pages 447–533 (cited on pages 4, 11, 13).

[BT01] Franz Baader and Stephan Tobies: 'The Inverse Method Implements the Automata Approach for Modal Satisfiability'. In *Proceedings of the International Joint Conference on Automated Reasoning IJCAR'01*. Edited by Rajeev Goré, Alexander Leitsch, and Tobias Nipkow. Volume 2083. Lecture Notes in Artificial Intelligence. Springer, 2001, pages 92–106 (cited on pages 52, 53, 66, 102).

[Ban22] Stefan Banach: 'Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales'. In *Fundamenta Mathematicae* **3**(1): 1922, pages 133–181 (cited on page 155).

[Bod65] VG Bodnarchuk: 'The metrical space of events. Part I'. In *Cybernetics and Systems Analysis* **1**(1): 1965, pages 20–24 (cited on page 47).

[BM96] Alexander Borgida and Deborah L. McGuinness: 'Asking Queries about Frames'. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*. 1996, pages 340–349 (cited on page 111).

[BWH05] Alexander Borgida, Thomas J. Walsh, and Haym Hirsh: 'Towards Measuring Similarity in Description Logics'. In *Proceedings of the 2005 International Workshop on Description Logics (DL2005), Edinburgh, Scotland, UK, July 26-28, 2005*. 2005 (cited on pages 5, 6, 85).

[Bra77] Ronald J. Brachman: 'What's in a concept: structural foundations for semantic networks'. In *International Journal of Man-Machine Studies* **9**(2): 1977, pages 127–152 (cited on page 2).

[Bra78] Ronald J. Brachman: 'Structured inheritance networks'. In. Quarterly Progress Report No. 1, BBN Report No. 3742: 1978, pages 36–78 (cited on page 2).

[BL84] Ronald J. Brachman and Hector J. Levesque: 'The Tractability of Subsumption in Frame-based Description Languages'. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence*. AAAI'84. Austin, Texas: AAAI Press, 1984, pages 34–37 (cited on pages 3, 77).

[BL87] Ronald J. Brachman and Hector J. Levesque: 'Expressiveness and tractability in knowledge representation and reasoning'. In. Volume 3. Feb. 1987, pages 78–93 (cited on page 6).

[Bra04] Sebastian Brandt: 'Polynomial Time Reasoning in a Description Logic with Existential Restrictions, GCI Axioms, and—What Else?' In *Proc. of the 16th Eur. Conf. on Artificial Intelligence (ECAI 2004)*. IOS Press, 2004, pages 298–302 (cited on page 6).

[BKT02] Sebastian Brandt, Ralf Küsters, and Anni-Yasmin Turhan: 'Approximation and Difference in Description Logics'. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*. 2002, pages 203–214 (cited on page 6).

[BHS87] Hans-Jürgen Bürckert, Alexander Herold, and Manfred Schmidt-Schauß: 'On equational theories, unification and decidability'. In *Rewriting Techniques and Applications*. Edited by Pierre Lescanne. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pages 204–215 (cited on page 25).

[Cou83] Bruno Courcelle: 'Fundamental properties of infinite trees'. In *Theoretical Computer Science* **25**(2): 1983, pages 95–169 (cited on page 51).

[dFE05] Claudia d'Amato, Nicola Fanizzi, and Floriana Esposito: 'A Semantic Similarity Measure for Expressive Description Logics'. In *Proc. of Convegno Italiano di Logica Computazionale (CILC05)*. Edited by A. Pettorossi. 2005 (cited on page 85).

[DG84] William F. Dowling and Jean Gallier: 'Linear-time Algorithms for Testing the Satisfiability of Propositional Horn Formulae'. In *Journal of Logic Programmming* **1**(3): 1984, pages 267–284 (cited on page 15).

[DKR08] Manfred Droste, Werner Kuich, and George Rahonis: 'Multi-Valued MSO Logics Over Words and Trees'. In *Fundam. Inf.* **84**(3,4): Dec. 2008, pages 305–327 (cited on page 74).

[DK06] Manfred Droste and Dietrich Kuske: 'Skew and infinitary formal power series'. In *Theoretical Computer Science* **366**(3): 2006, pages 199–227 (cited on pages 58, 64).

[DM10] Manfred Droste and Ingmar Meinecke: 'Describing Average- and Longtime-Behavior by Weighted MSO Logics'. In *Mathematical Foundations of Computer Science 2010*. Edited by Petr Hliněný and Antonín Kučera. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pages 537–548 (cited on page 62).

[DR09] Manfred Droste and George Rahonis: 'Weighted automata and weighted logics with discounting'. In *Theoretical Computer Science* **410**(37): 2009, pages 3481–3494 (cited on pages 58, 59).

[Eck17] Andreas Ecke: 'Quantitative Methods for Similarity in Description Logics'. PhD thesis. PhD thesis. Dresden University of Technology, Germany, 2017 (cited on pages 6, 85).

[EPT15] Andreas Ecke, Rafael Peñaloza, and Anni-Yasmin Turhan: 'Similarity-based Relaxed Instance Queries'. In *Journal of Applied Logic* **13**(4, Part 1): 2015. Special Issue for the Workshop on Weighted Logics for AI 2013, pages 480–508 (cited on pages 6, 85, 86).

[FK15] Sean A. Fulop and David Kephart: 'Topology of language classes.' In *The 14th meeting on the mathematics of language. Proceedings of the meeting, MoL 14, Chicago, IL, USA, July 25–26, 2015*. Stroudsburg, PA: Association for Computational Linguistics, 2015, pages 26–38 (cited on pages 47, 48).

[FV09] Zoltán Fülöp and Heiko Vogler: 'Weighted Tree Automata and Tree Transducers'. In *Handbook of Weighted Automata*. Edited by Manfred Droste, Werner Kuich, and Heiko Vogler. Springer Berlin Heidelberg, 2009, pages 313–403 (cited on page 56).

[GJ90] Michael R. Garey and David S. Johnson: *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990 (cited on pages 35, 118, 119).

[HS85] David Harel and Rivi Sherman: 'Propositional Dynamic Logic of Flowcharts'. In *Information and Control* **64**(1-3): 1985, pages 119–135 (cited on page 122).

[HLP08] Frank van Harmelen, Vladimir Lifschitz, and Bruce W. Porter, editors: *Handbook of Knowledge Representation*. Volume 3. Foundations of Artificial Intelligence. Elsevier, 2008 (cited on page 2).

[Hod97] Wilfrid Hodges: *A Shorter Model Theory*. New York, NY, USA: Cambridge University Press, 1997 (cited on page 130).

[HK03] Markus Holzer and Martin Kutrib: 'Nondeterministic descriptional complexity of regular languages'. In *International Journal of Foundations of Computer Science* **14**(06): 2003, pages 1087–1102 (cited on page 45).

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman: *Introduction to Automata Theory, Languages, and Computation (3rd Edition) (The Cinderella Book)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006 (cited on page 43).

[Jac12] Paul Jaccard: 'THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1'. In *New Phytologist* **11**(2): 1912, pages 37–50 (cited on page 47).

[Jan06] Krzysztof Janowicz: 'Sim-DL: Towards a Semantic Similarity Measurement Theory for the Description Logic $\mathcal{ALCNR}$ in Geographic Information Retrieval'. In *Proceedings of the 2006 International Conference on On the Move to Meaningful Internet Systems: AWeSOMe, CAMS, COMINF, IS, KSinBIT, MIOS-CIAO, MONET - Volume Part II*. OTM'06. Montpellier, France: Springer-Verlag, 2006, pages 1681–1692 (cited on page 6).

[JS87] Brigitte Jaumard and Bruno Simeone: 'On the Complexity of the Maximum Satisfiability Problem for Horn Formulas'. In *Inf. Process. Lett.* **26**(1): 1987, pages 1–4 (cited on page 16).

[KN86] Deepak Kapur and Paliath Narendran: 'NP-Completeness of the Set Unification and Matching Problems'. In *Proceedings of the 8th International Conference on Automated Deduction*. Edited by Jörg H. Siekmann. Volume 230. Lecture Notes in Computer Science. Oxford, UK: Springer, 1986, pages 489–495 (cited on page 31).

[KN92] Deepak Kapur and Paliath Narendran: 'Complexity of Unification Problems with Associative-Commutative Operators'. In *Journal of Automated Reasoning* **9**(2): Oct. 1992, pages 261–288 (cited on pages 5, 7, 11, 13–15, 24, 31).

[Kep05] David E Kephart: 'Topology, morphisms, and randomness in the space of formal languages'. PhD thesis. PhD thesis. University of South Florida, Tampa FL, USA, 2005 (cited on pages 46–48).

[Kle56] Stephen Cole Kleene: 'Representation of events in nerve nets and finite automata'. In *Automata Studies*. Edited by Claude Shannon and John McCarthy. Princeton, NJ: Princeton University Press, 1956, pages 3–41 (cited on page 44).

[Kle02] Stephen Cole Kleene: *Mathematical logic*. Reprint of the 1967 original. Mineola, NY: Dover Publications, 2002 (cited on page 73).

[Kno51] Konrad Knopp: *Theory and Application of Infinite Series*. New York: Hafner Publishing Company, 1951 (cited on page 57).

[Kre78] E. Kreyszig: *Introductory Functional Analysis With Applications*. Wiley Classics Library. John Wiley & Sons, 1978 (cited on page 155).

[KN06] Manfred Kudlek and Benedek Nagy: 'Distances of formal languages.' In *PU.M.A., Pure Math. Appl.* **17**(3-4): 2006, pages 349–357 (cited on pages 46–48, 62).

[Kun07] Michal Kunc: 'What Do We Know About Language Equations?' In *Proc. of the 11th International Conference on Developments in Language Theory (DLT 2007)*. Edited by Tero Harju, Juhani Karhumäki, and Arto Lepistö. Volume 4588. Lecture Notes in Computer Science. Springer-Verlag, 2007, pages 23–27 (cited on page 143).

[Küs01] Ralf Küsters: *Non-standard Inferences in Description Logics*. Volume 2100. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2001 (cited on page 86).

[Leh12] Karsten Lehmann: 'A Framework for Semantic Invariant Similarity Measures for ELH Concept Descriptions'. Diploma Thesis. Dresden, Germany: Technische Universität Dresden, 2012 (cited on page 90).

[LT12] Karsten Lehmann and Anni-Yasmin Turhan: 'A Framework for Semantic-based Similarity Measures for $\mathcal{ELH}$-Concepts'. In *Proc. of the 13th Eur. Conf. on Logics in Artificial Intelligence (JELIA'2012)*. Volume 7519. Lecture Notes in Computer Science. Springer, 2012, pages 307–319 (cited on pages 5, 6, 85, 86, 89, 90).

[MR09] Eleni Mandrali and George Rahonis: 'Recognizable Tree Series with Discounting'. In *Acta Cybernetica* **19**(2): 2009, pages 411–439 (cited on page 58).

[Mar96] Claude Marché: 'Normalized Rewriting: an Alternative to Rewriting modulo a Set of Equations'. In *Journal of Symbolic Computation* **21**(3): 1996, pages 253–288 (cited on page 12).

[MIM17] Joao Marques-Silva, Alexey Ignatiev, and Antonio Morgado: 'Horn Maximum Satisfiability: Reductions, Algorithms and Applications'. In *Progress in Artificial Intelligence*. Edited by Eugénio Oliveira, João Gama, Zita Vale, and Henrique Lopes Cardoso. Cham: Springer International Publishing, 2017, pages 681–694 (cited on page 16).

[Mun00] J.R. Munkres: *Topology*. Featured Titles for Topology Series. Prentice Hall, Incorporated, 2000 (cited on page 155).

[Nav01] Gonzalo Navarro: 'A Guided Tour to Approximate String Matching'. In *ACM Comput. Surv.* **33**(1): Mar. 2001, pages 31–88 (cited on page 46).

[Neb90] Bernhard Nebel: 'Terminological reasoning is inherently intractable'. In *Artificial Intelligence* **43**: 2 May 1990, pages 235–249 (cited on pages 3, 7).

[PRZ16] Jeff Z. Pan, Yuan Ren, and Yuting Zhao: 'Tractable approximate deduction for OWL'. In *Artificial Intelligence* **235**: 2016, pages 95–155 (cited on page 6).

[Pap94] Christos H. Papadimitriou: *Computational complexity*. Addison-Wesley, 1994 (cited on page 43).

[Pen15] Maximilian Pensel: 'An automata based approach for subsumption w.r.t. general concept inclusions in the description logic $\mathcal{FL}_0$'. Master's thesis. Chair for Automata Theory, TU Dresden, Germany, 2015 (cited on pages 2, 7, 50, 78, 79, 124, 128, 129).

[Per90] Dominique Perrin: 'Handbook of Theoretical Computer Science (Vol. B)'. In. Edited by Jan van Leeuwen. Cambridge, MA, USA: MIT Press, 1990. Chapter Finite Automata, pages 1–57 (cited on page 43).

[PKB55] James W. Perry, Allen Kent, and Madeline M. Berry: 'Machine literature searching X. Machine language; factors underlying its design and development'. In *American Documentation* **6**(4): 1955, pages 242–254 (cited on page 1).

[Pra80] Vaughan R. Pratt: 'A Near-Optimal Method for Reasoning About Action'. In *Journal of Computer and System Sciences* **20**(2): 1980, pages 231–255 (cited on page 122).

[Rab72] Michael Oser Rabin: *Automata on Infinite Objects and Church's Problem*. Boston, MA, USA: American Mathematical Society, 1972 (cited on page 52).

[RS15] T. Racharak and B. Suntisrivaraporn: 'Similarity measures for $\mathcal{FL}_0$ concept descriptions from an automata-theoretic point of view'. In *6th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES)*. 2015, pages 1–6 (cited on pages 6, 86, 87).

[Rah07] George Rahonis: 'Weighted Muller Tree Automata and Weighted Logics'. In *J. Autom. Lang. Comb.* **12**(4): 2007, pages 455–483 (cited on pages 56, 57, 59).

[Rud87] Walter Rudin: *Real and Complex Analysis, 3rd Ed.* New York, NY, USA: McGraw-Hill, Inc., 1987 (cited on pages 47, 90).

[Sch91] Klaus Schild: 'A Correspondence Theory for Terminological Logics: Preliminary Report'. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'91. Sydney, New South Wales, Australia: Morgan Kaufmann Publishers Inc., 1991, pages 466–471 (cited on pages 77, 122).

[Sch99] Alexander Schrijver: *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1999 (cited on page 71).

[She87] Roger N Shepard: 'Toward a universal law of generalization for psychological science'. In *Science* **237**(4820): 1987, pages 1317–1323 (cited on page 85).

[Sie89] Jörg H. Siekmann: 'Unification theory: A Survey'. In *Journal of Symbolic Computation* **7**(3): 1989, pages 207–274 (cited on pages 4, 11).

[Sun13] Boontawee Suntisrivaraporn: 'A Similarity Measure for the Description Logic $\mathcal{EL}$ with Unfoldable Terminologies'. In *5th Int. Conf. on Intelligent Networking and Collaborative Systems*. IEEE, 2013, pages 408–413 (cited on pages 6, 85, 86).

[Tho90] Wolfgang Thomas: 'Automata on Infinite Objects'. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. The MIT Press, 1990, pages 133–192 (cited on page 51).

[VW86] Moshe Y. Vardi and Pierre Wolper: 'Automata-theoretic techniques for modal logics of programs'. In *Journal of Computer and System Sciences* **32**(2): 1986, pages 183–221 (cited on page 52).

[VW94] Moshe Y. Vardi and Pierre Wolper: 'Reasoning about Infinite Computations'. In *Information and Computation* **115**(1): 1994, pages 1–37 (cited on page 52).

[Via77] Victor Vianu: 'The Bodnarchuk Metric Space of Languages and the Topology of the Learning Space'. In *Mathematical Foundations of Computer Science 1977, Proceedings*. 1977, pages 537–542 (cited on page 47).

[Yu97] Sheng Yu: 'Regular Languages'. In *Handbook of Formal Languages (1)*. 1997, pages 41–110 (cited on page 43).

[YZS94] Sheng Yu, Qingyu Zhuang, and Kai Salomaa: 'The state complexities of some basic operations on regular languages'. In *Theoretical Computer Science* **125**(2): 1994, pages 315–328 (cited on page 45).

Appendix: Metric Topology

In this appendix we recall some basic notions from Metric Topology. Formal proofs and discussion on the results we will now mention can be found in any book on the subject, for example [Kre78; Mun00].

Given a set X, a *metric* (or *distance*) on X is a mapping $d : X \times X \to [0, \infty)$ that satisfies the properties:

$(M1)$ $d(a,b) = 0 \iff a = b$

$(M2)$ $d(a,b) = d(b,a)$

$(M3)$ $d(a,c) \leqslant d(a,b) + d(b,c)$

In this case, (X,d) is called a *metric space*. A useful metric on $\mathbb{R}^k$ is the Chebyshev distance, d_∞, which for $\vec{x} = (x_1, \ldots, x_k)$, $\vec{y} = (y_1, \ldots, y_k) \in \mathbb{R}^k$ is defined as

$$d_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1,\ldots,k} |x_i - y_i|.$$

Given a metric space (X,d), a sequence (a_n) of elements of X is said to *converge* to $a \in X$ (written $a_n \xrightarrow{d} a$) if for every $\varepsilon > 0$ there is an $n_0 \in \mathbb{N}$ s.t. $d(a_n, a) < \varepsilon$ for every $n \geqslant n_0$. For a sequence $((a_1^n, \ldots, a_k^n))_n$ of elements of $\mathbb{R}^k$, we have that $(a_1^n, \ldots, a_k^n) \xrightarrow{d_\infty} (a_1, \ldots, a_k)$ iff $a_i^n \xrightarrow{d_\infty} a_i$ for every $i = 1, \ldots, k$. A sequence (a_n) is called a *Cauchy* sequence, if for every $\varepsilon > 0$, there exists an $n_0 \in \mathbb{N}$, s.t. for every $m, n \geqslant n_0$ it holds that $d(a_n, a_m) < \varepsilon$. A metric space (X,d) is called *complete*, if every Cauchy sequence converges to a point in X. It is well known that the metric space $(\mathbb{R}^k, d_\infty)$ is complete [Kre78].

Definition A.1. *Given a metric space (X,d) a function $f : X \to X$ is called a* contraction*, if there is a $\lambda \in (0,1)$ such that $d(f(x), f(y)) \leqslant \lambda d(x,y)$ for any $x, y \in X$.* ♢

Theorem A.2 (Banach's Fixed Point Theorem [Ban22; Kre78]). *Let (X,d) be a complete metric space and a function $f : X \to X$ be a contraction. Then there exists a unique fixed point, i.e., $p \in X$ such that $f(p) = p$.*

Furthermore, let $(X,d), (Y,d')$ be metric spaces. A function $f : X \to Y$ is called *continuous*, if for every sequence (a_n) of X, it holds that

$$a_n \xrightarrow{d} a \implies f(a_n) \xrightarrow{d'} f(a).$$

Finally, we provide the formal definition of the *infimum* of a set of real numbers, which will be needed in the proofs.

Definition A.3. *Given a set of real numbers S, we say that p is the* infimum *of S, and denote this by $p = \inf S$ if the following two conditions hold:*

(*a*) $p \leqslant s$ *for all* $s \in S$*, i.e.* p *is a lower bound of* S*,*

(*b*) *for all* $\varepsilon > 0$*, there is an* $s \in S$ *such that* $s < p + \varepsilon$. ♢

Note that this means that if $p = \inf S$, there exists a sequence (s_n) of elements of S, s.t. $s_n \xrightarrow{d_\infty} p$. Furthermore, if f is continuous and monotone, then f distributes over infimum in the following sense.

Lemma A.4. *Assume that* $f : \mathbb{R}^n \to \mathbb{R}$ *is continuous and monotone, and let* $I_1, \ldots, I_n \subseteq \mathbb{R}$*. Then*

$$\inf_{(x_1,\ldots,x_n)\in I_1\times\cdots\times I_n} f(x_i,\ldots,x_n) = f\left(\inf_{x_1\in I_1} x_1,\ldots,\inf_{x_n\in I_n} x_n\right).$$

List of Symbols

Description Logics

Languages, Trees, and Automata

Mathematical symbols

Unification Theory

Index