FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automata-based Reasoning for Decidable Logics with Data Values

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

Nadia Labai, MSc
Matrikelnummer 1529069

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.techn. Magdalena Ortiz
Zweitbetreuung: Privatdoz. Dr.techn. Mantas Šimkus

Diese Dissertation haben begutachtet:

Anni-Yasmin Turhan

Giuseppe De Giacomo

Wien, 6. Mai 2021

Nadia Labai

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automata-based Reasoning for Decidable Logics with Data Values

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Nadia Labai, MSc
Registration Number 1529069

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dr.techn. Magdalena Ortiz
Second advisor: Privatdoz. Dr.techn. Mantas Šimkus

The dissertation has been reviewed by:

Anni-Yasmin Turhan

Giuseppe De Giacomo

Vienna, 6th May, 2021

Nadia Labai

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Nadia Labai, MSc
Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. Mai 2021

Nadia Labai

Acknowledgements

None of the work in this dissertation would have seen the light of day if it wasn't for my advisor, Magdalena Ortiz. Magdalena was an infinite source of encouragement, guidance, and support from the moment we began working together. Always calm, always confident that whatever it is – we will figure it out, even when I doubted it. I am grateful for her flexibility with the course of this doctorate, which was expressed in so many ways including the very topic of this dissertation. Magdalena, I am fortunate to have had you as a mentor through this time of my life both academically and personally. Thank you for that. I am thankful to have had Mantas Šimkus as my co-advisor. Mantas, the openness of our back-and-forths is something I will always cherish. I would also like to thank Magdalena and Mantas for their continuous help and empathy as my husband and I became parents while trying to navigate our doctorates. I could not ask for more.

I gratefully acknowledge the generous funding by the Austrian Science Fund (FWF) projects P30360, P30873, and W1255.

I thank Tomer Kotek for letting my intuition run free on the whiteboard. It was a pleasure working with you. Warm thanks go to Martin Homola for hosting me at Comenius University in Bratislava and to the KR group for creating such fond memories during my time there. I'm grateful to have had the opportunity to work with and learn from the late Helmut Veith during my first year here. Helmut's passion for collaboration is something I will never forget. I'd like to express my appreciation of the relentless efforts made by Anna Prianichnikova to ensure the LogiCS DK is a success. Thank you to Juliane Auerböck, Beatrix Forsthuber, Eva Nedoma, and Toni Pisjak for always finding a solution to any issue. I'd like to express my sincere gratitude to Giuseppe De Giacomo and Anni-Yasmin Turhan for their helpful feedback on this dissertation.

Big thanks go to my office-mates and coffee-mates at the LogiCS for fun lighthearted chats: Shqiponja Ahmetaj, Labinot Bajraktari, Katalin Fazekas, Benjamin Kiesl, Jens Pagel, and Zeynep G. Saribatur.

Thank you to Betsy Blankrot and Leon Blankrot for your support and affection. Thank you to Janos Makowsky and Masha Yelenevskaya for your long-lasting friendship and honest advice. Thank you to Inbal Ipenberg for being someone I can talk to, always.

Lastly, I thank my little family. Boaz, thank you for the hundreds (if not thousands) of hours spent listening to me talk about all aspects of this work. Thank you for your faith in me and for your unwavering love during this endeavor, which seemed endless at times. I am finally here. Thank you to my son, Rafi, for being a truly wonderful part of my life. You've taught me so much without even trying.

Kurzfassung

Entscheidbare Logiken spielen eine wichtige Rolle bei der Wissensrepräsentation, der automatisierten Verifikation und der Datenbankverwaltung. Eine Gemeinsamkeit, die viele dieser Logiken verbindet, ist das Zwei-Variablen-Fragment der Prädikatenlogik erster Stufe (FO^2), das sich gut für den Umgang mit graph-strukturierten Daten eignet. Insbesondere umfasst FO^2 viele so genannte Description Logics (DLs), einer weit verbreiteten Familie von Formalismen für den Ausdruck von Hintergrundwissen, das genutzt werden kann, um implizite Verbindungen oder Inkonsistenzen in großen Informationsmengen zu finden. FO^2 ist auch als Grundlage von Verifikationslogiken zur Spezifizierung von Systemen nützlich, um ihr gewünschtes Verhalten und ihre gewünschten Eigenschaften sicherzustellen. Für Anwendungen in der realen Welt ist es jedoch fast immer erforderlich, so genannte konkrete Werte wie Messungen, Strings und Zeit adäquat zu modellieren. Eine solche Modellierung wird weder von DLs aufgrund ihrer abstrakten Natur noch von FO^2 unterstützt, da FO^2 zur Modellierung notwendige Eigenschaften wie Transitivität nicht ausdrücken kann. Daher wurden Anstrengungen unternommen, Datenwerte in DLs zu integrieren und FO^2 um Möglichkeiten zur Datenmodellierung unter Beibehaltung der Entscheidbarkeit zu erweitern.

Das übergreifende Ziel dieser Arbeit ist die Entwicklung von auf Automaten basierenden Techniken zur Schlussfolgerung in ausdrucksstarken entscheidbaren Logiken, die es erlauben, Datenwerte zu modellieren. Automatentheoretische Schlussfolgerungstechniken spielen in der Logik und Informatik eine zentrale Rolle und sind aus vielen Gründen attraktiv. Sie sind intuitiv und erlauben es, technische Details zu abstrahieren, die es erschweren können, die Ausdruckskraft einer Logik zu verstehen. Sie sind oft modular aufgebaut und können auf andere Formalismen übertragen werden und bieten so eine Grundlage für den Vergleich von Logiken. Darüber hinaus geben die Änderungen am Automatenmodell, die zur Erfüllbarkeitsprüfung ausdruckstärkerer Logiken vorgenommen werden, oft Aufschluss über deren Komplexität, da automaten-theoretische Lösungen für logische Schlussfolgerungsprobleme in der Regel eine optimale Worst-Case-Berechnungskomplexität aufweisen. In dieser Arbeit konzentrieren wir uns auf zwei verschiedene ausdrucksstarke Logiken mit Datenwerten und setzen Automaten zur Lösung der folgenden offenen Probleme ein.

- Zunächst betrachten wir eine DL mit dem Namen $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, die Einschränkungen unterstützt, die als Vergleiche von ganzen Zahlen angegeben werden. Die Entscheidbarkeit dieser Logik wurde über ein mächtiges Meta-Theorem gezeigt, das schwache

monadische Logik zweiter Stufe verwendet. Dieses Ergebnis bietet jedoch keine obere Komplexitätsschranke für das Problem. Indem wir einen Rabin-Baumautomaten für $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ konstruieren, erhalten wir eine enge ExpTime-Obergrenze für die Entscheidbarkeit des s.g. *concept satisfiability problem* in Bezug auf allgemeine TBoxen. Dies ist unseres Wissens nach die erste obere Komplexitätsschranke für die Schlussfolgerung mit allgemeinen TBoxen und diskreten konkreten Domänen. Unsere Ergebnisse liefern auch einige obere Schranken für Entscheidbarkeit und Komplexität verwandter Probleme in DLs mit numerischen Werten, einschließlich lang gesuchter konkreter Domänen über den reellen Zahlen, die mit Prädikaten ausgestattet sind, die erzwingen, dass einige Werte ganzzahlig oder natürlich sind.

- Zweitens betrachten wir die Logik $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$, deren endliches Erfüllbarkeitsproblem als ExpSpace-vollständig bekannt ist. Bei dieser Logik handelt es sich um ein Zwei-Variablen-Fragment der Prädikatenlogik erster Stufe, dessen Signatur eine lineare Ordnung, eine Quasiordung und deren Nachfolgerrelation, sowie eine beliebige endliche Anzahl von unären Prädikaten enthält. Die Quasiordnung modelliert auf natürliche Weise Datenwerte, die aus einer unendlichen Domäne stammen, und die lineare Ordnung ist für die Modellierung von *program traces* geeignet, so dass diese Logik ein guter Kandidat für Anwendungen in der Programmverifizierung ist. Um die Verbindung zwischen Logik und Automaten für diese Logik zu untersuchen, führen wir ein neuartiges Automatenmodell namens Pebble-Intervals Automata (PIA) ein, das $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ umfasst. Wir untersuchen die Leistungsfähigkeit von PIAs im Gegensatz zu anderen Automatenmodellen, und untersuchen das Leerheitsproblem und die Abschlusseigenschaften von PIAs. Wir erhalten auch einen automaten-theoretischen Beweis einer oberen Komplexitätsschranke für das endliche Erfüllbarkeitsproblem der Logik durch Reduktion auf einen PIA-Leerheitstest.

Abstract

Decidable logics play a major role in knowledge representation, automated verification, and database management. A common ground relating many of these logics is the two-variable fragment of First Order logic (FO^2), which is well-suited for handling graph-structured data. In particular, FO^2 encompasses many Description Logics (DLs) which are popular formalisms for expressing background knowledge that can be leveraged to find implicit connections or inconsistencies in large amounts of information. FO^2 is also useful as the basis of verification logics for specifying systems to ensure their desired behavior and properties. However, real-world applications nearly always require us to adequately model so-called concrete values such as measurements, strings, and time. Such modeling abilities are not supported by DLs due to their abstract nature, nor by FO^2 as it cannot express necessary properties such as transitivity. As a result, efforts have been made to incorporate data values into DLs, and to extend FO^2 with means for modeling data while maintaining decidability.

The overarching goal of this thesis is to develop techniques based on automata for reasoning in rich decidable logics that allow to model data values. Automata-theoretic reasoning techniques play a central role in logic and computer science, and are attractive for many reasons. They are intuitive and allow one to abstract away technical details that may muddy the waters when trying to understand the expressiveness of a logic. They are often modular and can be transferred across settings, offering a basis for comparison between logics. Furthermore, the adjustments made to accommodate new capabilities often provide insight into their computational cost, since automata-theoretic solutions to logical reasoning problems typically display optimal worst-case computational complexity. In this thesis, we concentrate on two different expressive logics with data values, and employ automata for solving the following open problems.

- First, we consider a DL called $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ that supports constraints given as comparisons of integer numbers. Decidability of this logic has been established via a powerful meta-theorem that relies on Weak Monadic Second Order logic, however, this result provides no upper complexity bounds for the problem. By constructing a Rabin tree automaton for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, we obtain a tight ExpTime upper bound for deciding concept satisfiability w.r.t. general TBoxes. This is, to our knowledge, the first complexity upper bound for reasoning with general TBoxes and discrete

concrete domains. Our results also yield some decidability and complexity upper bounds for related problems in DLs with numeric values, including long sought-after concrete domains over the real numbers equipped with predicates that enforce some values to be integer or natural.

- Second, we consider $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$, whose finite satisfiability problem is known to be ExpSpace-complete. This logic is the two variable-fragment of First Order logic equipped with a linear order, a preorder and its successor relation, and any finite number of unary predicates. The preorder nicely models data values that may range over an infinite domain, and the linear order is appropriate for modeling program traces, making this logic a good candidate for applications involving program verification. To study the logic-automata connection for this logic, we introduce a novel automata model called Pebble-Intervals automata (PIA) that is designed to encompass $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$. We study the power of PIAs in contrast to other automata models, and investigate its emptiness problem and its closure properties. We also obtain an automata-theoretic proof of the upper bound for the finite satisfiability problem of the logic by reducing it to a PIA emptiness test.

Contents

CHAPTER 1

Introduction

The idea of using logical formulas for representing facts, rules, and knowledge about the world, and mechanizing logical deduction in order to draw correct inferences from such knowledge, has always been at the very core of computer science. In fact, it can be traced much further than modern computation; Frege attempted to provide a formal logical foundation to mathematics, which followed Boole's separation of the symbols representing concepts from their manipulation and validity in the 1800's. Earlier still, in the late 17th century Leibniz extensively studied the idea of representing complex thoughts as the result of manipulating symbols representing atomic ideas. This common thread of thoughts as symbols goes all the way back to 350 BC, we have Aristotle's writings on sound deduction in Prior Analytics.

Thus it does not come as a surprise that this dream played a particularly influential role in the birth of Artificial Intelligence (AI), whose fundamental goals include the creation of so-called commonsense knowledge and reasoning. The Dartmouth Summer Research Project on Artificial Intelligence, often quoted as the founding event of AI as a field, was proposed by John McCarthy [1] based on the conjecture that "every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it". As mentioned, early attempts to achieve this goal were based on logic, and logical inference remains a fundamental tool for inference in AI. However, deploying this approach successfully in practice calls for logical formalisms with a decidable satisfiability problem, that is, where it is decidable whether a given formula in the logic has a satisfying assignment – a model.

As mathematicians were studying the idea of deriving logical proofs in a mechanical manner in the early 20th century, particular attention was given to First Order logic (FO). FO is capable of axiomatizing various branches of mathematics using finitely many axioms, which together with a formal proof system allows us to rephrase mathematical claims as logical formulas, and to rephrase the question of whether the claim holds given some assumptions as the question of whether a logical formula is valid in our proof system.

If we were careful enough to select a sound (meaning we can only infer true things) and complete (meaning we can infer everything that is true) system, this is equivalent to asking whether the formula is *derivable* from the axioms in our system. That is, whether there is a sequence of applicable operations that take us from our axioms to the claim.

Naturally, such proof systems for FO would be very desirable, and in 1928 Hilbert and Ackerman published a sound system [2] which Gödel proved complete [3]. However, the ability to derive every true FO sentence is quite different from answering whether a *given* FO sentence is true, and Hilbert established this problem - the decision problem ("das Entscheidungsproblem") - of FO as a central problem in mathematics. Unfortunately, this questions has been answered negatively by Church [4], Turing [5], and Trakhtenbrot [6], but their answers did spur immense research efforts aimed at obtaining decidability by imposing restrictions on the formulas. Among these are restrictions on the prenex form of formulas, which have a wealth of negative and positive results. For example, the Gurevich-Maslov-Orevkov class, which consists of FO formulas with the prefix $\exists^\star\forall\exists^\star$ and without equality, is decidable in exponential time [7]. On the other hand, satisfiability of FO formulas with the prefix $\forall\exists\forall$ without equality is undecidable [8]. See e.g. [9] for many more results on prenex classes of FO.

The Guarded Fragment (GF) of FO [10] is another decidable fragment of FO, where quantifiers are relativized (guarded) by atomic formulas. Roughly, the guarded fragment is inductively defined by saying atomic formulas are guarded, boolean combinations of guarded formulas are guarded, and formulas of the form $\exists\bar{v}(\varphi \wedge \psi)$ and $\forall\bar{v}(\varphi \rightarrow \psi)$ are guarded if φ is atomic and all the free variables in ψ appear in φ. There is no restriction on the variables $\bar{v}$. The GF enjoys a 2ExpTime-complete satisfiability problem [11] in the general case, and is ExpTime-complete when the arity of relations is bounded [11].

Another restriction is on the number of different variables in the formula. Here there is far less room to maneuver on syntax alone, since already the three-variable fragment of FO is undecidable as it subsumes another undecidable fragment [8]. However, the two-variable fragment FO^2 is decidable as it enjoys a finite bounded-size model property, meaning any satisfiable sentence in FO^2 has a model of some known bounded size. Mortimer was the first to prove this property with a double-exponential bound on the model [12], and Grädel et al. to improved it to obtain NExpTime-completeness [13] for the problem.

FO^2 has since emerged as a common ground for many logics employed for AI tasks. Like the GF, FO^2 has many extensions where new relations whose interpretations are assumed are added, such as linear orders or equivalence relations. For example, FO^2 with one equivalence relation and one linear order is decidable [14], but a similar setting with an equivalence relation and a transitive relation is undecidable [15]. Another important fragment is C^2, which is FO^2 with quantifiers such as $\exists^{<k}$, $\exists^{=k}$, etc. Extensions that are based on FO^2 are often well suited for modeling (labeled) graphs, trees, and strings, and so they are particularly useful in Computer Science applications.

In particular, FO^2 encompasses many Description Logics (DLs), which are a family of logics for representing knowledge and reasoning about application domains in a precise

way. DLs can be used to formalize ontologies for medicine (SNOMED CT[1]), finance (FIBO[2]), and the Semantic Web (OWL[3]), where by 'ontology' we mean an explicit specification of concepts, properties, and relationships in our domain of interest. DLs have been a rapidly developing area of research that evolved from semantic networks and frame systems, with the aim of providing high-level and precise descriptions of the interest domain while delivering reasoning services. Thus the choice of DL depends on the reasoning task we wish to perform, and typically strikes a balance between expressiveness and computational tractability. Knowledge representation is achieved using concepts, which are classes of elements sharing common properties, and roles, which are binary relations connecting these elements. Through concepts and roles, one can define axioms describing the domain (in what is called a TBox), and extract new insights using the reasoning services offered by the DL. For example, it would not be surprising to find something along the lines of:

$$\exists \mathsf{nativeTo.Australia} \sqcap \exists \mathsf{eats.EucalyptusLeaves}$$

in a description of koalas (Figure 1.1). Here, we have nativeTo and eats as role names, and Australia and EucalyptusLeaves as concept names. A TBox may contain axioms such as:

$$\begin{aligned} \mathsf{Koala} &\sqsubseteq \exists \mathsf{nativeTo.Australia} \sqcap \exists \mathsf{eats.EucalyptusLeaves} \\ \mathsf{Koala} &\sqsubseteq \mathsf{Herbivore} \\ \mathsf{Herbivore} \sqcap \exists \mathsf{eats.Meat} &\equiv \bot \end{aligned}$$

Where the last axiom states that herbivores do not eat meat. The first axiom can be expressed in the 2-variable fragment as:

$$\forall x.koala(x) \rightarrow \Big(\exists y.\, (nativeTo(x,y) \wedge Australia(y)) \\ \wedge \exists y.\, (eats(x,y) \wedge EucalyptusLeaves(y)) \Big)$$

We say that the concepts appearing on the right hand side of an axiom are *used* by the concept appearing on the left hand side. Concept satisfiability w.r.t a TBox is the problem of determining whether there can be a member of a certain concept while respecting the axioms in the TBox. For example, can there be a koala that eats meat? I.e., is the concept $\mathsf{Koala} \sqcap \exists \mathsf{eats.Meat}$ satisfiable w.r.t to our TBox?

In addition to description logics, FO^2 is also related to logics used in automated verification; besides the well-known subsumption of Modal logic by FO^2, FO^2 also serves as

[1] http://www.snomed.org/
[2] https://spec.edmcouncil.org/fibo/
[3] https://www.w3.org/OWL/
* creativecommons.org/licenses/by-sa/3.0/

Figure 1.1: A koala eating eucalyptus leaves. Photo by John 'Sheba Also'. CC-BY-SA-3.0⋆

a basis for logics used in the automated verification of various properties of programs, including safety (e.g. in a traffic light, the red and green light are never simultaneously on), liveness (every red light is eventually followed by a green light), and temporal properties (the light is green infinitely often). There has also been work in combining FO^2 based approaches with DLs for the automated verification of programs with dynamic data structures [16], as well as the verification of integrity constraint preservation [17], and reasoning about dynamically allocated memory [18].

In the context of real-world applications, it is nearly always the case that the domain elements have measurements of length, time, weight, or dates associated with them. Thus it is imperative for the logical formalism to properly model numeric data. However, such modeling capabilities are not natively supported by DLs nor by FO^2. DLs speak of the world using abstract concepts and roles, so it is not clear how to model the fact that koalas tend to weigh between 4 to 15 kg [19]. One possibility is a statement such as

$$\mathsf{Koala} \sqsubseteq \exists \mathsf{weighs}.\mathsf{MoreThan4} \sqcap \exists \mathsf{weighs}.\mathsf{LessThan15}$$

but it does not truly reflect the between-ness we wish to capture, as we can easily satisfy nonsensical concepts such as $\mathsf{LessThan4} \sqcap \mathsf{MoreThan15}$. As another example, the usual linear order of the integer or natural numbers is not reflected by DL concepts or roles, and FO^2 inherently lacks the ability to express transitivity, as it would contradict its finite model property. The need to support reasoning in the presence of numeric values has been recognized in both these arenas.

Since the early days of DLs, substantial effort was directed to enriching DLs with modeling capabilities for numeric domains while maintaining decidability. Such DLs are usually dubbed *DLs with concrete domains*, with a typical concrete domain being comprised of the domain elements (for example, the real numbers $\mathbb{R}$) and predicates over those elements (for example, the binary 'smaller than' relation $<$). The first DLs with concrete domains were introduced by Baader and Hanschke [20], where concrete values are connected via feature paths, meaning each connection between two elements is functional (for example, 'date of birth' can relate persons with dates in a functional way). They showed that satisfiability of concepts in such DLs, that is, whether there can be an element in the universe that is described by that concept is decidable for concrete domains $\mathcal{D}$ that are *admissible*. Admissible concrete domains are those where satisfiability of conjunctions of predicates from $\mathcal{D}$ is decidable, and its predicates are closed under negation. Generalizations of this result and tight complexity bounds for specific settings were obtained in the following years. For example, concept satisfiability is PSpace-complete under certain assumptions [21], and introducing stronger modeling capabilities increases the complexity to NExpTime or even results in undecidability [22]. A summary of key results can be found in the survey paper [23].

In automated verification, one commonly faces the need to model the presence of data from infinite domains. For example, numbers in program variables, databases, and timing in memory access of concurrent processes. These and similar settings can be represented using data words, which are strings where each position carries a data value from some infinite domain, in addition to carrying a letter from a finite alphabet, as is familiar. This motivated the exploration of extensions of FO^2 which can express properties of data words using built-in relations that are not axiomatizable in FO^2. Namely, linear orders, preorders, and equivalence relations have received substantial attention as candidates for modeling data in general and for reasoning about data words [24, 25, 26, 27, 28, 29].

1.1 Motivation and goals

In this thesis, we explore automata for these kinds of decidable logics with data modeling capabilities. The automata-logic connection is fundamental in Computer Science. Classical results by Schützenberger [30], McNaughton and Papert [31] relate first order logic and star-free languages, which are those recognized by counter-free automata. The Büchi-Elgot-Trakhtenbrot Theorem [32, 33, 34] relates definability in Monadic Second Order logic with regularity, and similar results have been discovered for graphs, see e.g. [35]. Automata-theoretic techniques have been applied to the verification of safety, liveness, and temporal properties of programs, which naturally translate to automata problems. In addition, automata can be used to verify relationships between systems, such as equivalence or refinement which are important for query optimization, as these naturally translate to language inclusion problems, see [36].

Finally, automata have played an important role in solving DL reasoning tasks, especially in transferring techniques between settings [37]. Tree automata in particular are often

used to answer consistency questions; i.e. whether a concept is satisfiable given a certain TBox, and subsumption questions, but automata have also been used for query answering over knowledge bases which include, in addition to a TBox, a set of assertions dubbed an ABox (these can state facts about *particular* koalas).

Beyond their historical importance, automata-theoretic techniques are especially attractive for exploring the expressiveness of *new* logical formalisms, as they offer a more intuitive 'behavioral' understanding of the logic without the burden of technical syntactic details which can get hairy quickly once more involved concepts are considered. Another aspect which encourages the use of automata for logic is that they are modular and can be transferred from setting to setting with relative ease. This fits well with the fact that DLs are often tailored to the problem at hand by tuning their constructors toward some sweet-spot that balances expressiveness and complexity. Finally, both for DLs and in automated verification, reductions to automata problems typically preserve optimal worst-case complexity, which allows one to gain insight into the computational cost of adding new features to the logic.

As mentioned, *the main goal of the thesis is to use automata for reasoning in expressive decidable logics that model data values, and to explore whether they offer the usual advantages described above.* As we must naturally limit our scope, we chose two different expressive decidable logics and two different problems to solve using automata.

1.1.1 Integer constraints in the logic $\mathcal{ALCF}$

Motivated by the importance of incorporating numeric data to knowledge representation, we continue the vast line of research involving DLs and concrete domains and consider the description logic $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, which was introduced by Carapelle and Turhan [38]. This logic allows one to levy constraints on registers associated with logical elements along paths. Oftentimes, the incorporation of concrete domains into a DL quickly results in undecidability, which is then regained by imposing limitations either on the logic or on the concrete domain. The choice of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ is motivated by the fact that it does not experience any of the three limitations usually encountered in the classical setting, which can be roughly classified like so:

- TBox prohibition or acyclicity
- Functionality of paths to the concrete domain
- Denseness of the concrete domain

We expand on each of these next.

TBox prohibition or acyclicity It is very well-known that general concept inclusions easily lead to undecidability in the classical setting, as witnessed by the progressively weaker extensions of $\mathcal{ALC}$ with concrete domains that are undecidable [39, 40, 41]. Often,

only *acyclic* TBoxes are allowed, which are TBoxes whose axioms do not have a concept which ultimately uses itself (either directly or via intermediate concepts). This is a major hindrance, since this type of self-reference is fairly common in real-world settings where elements of a class have relationships with other elements in the class. The desire to reflect such relationships motivates the choice of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, as there is no need to restrict the TBox in order to maintain decidability. That is not to say that general TBoxes are not ever supported in the classical setting, see for example [42] and [41], but in those instances one still encounters restrictions on the mechanisms that grant access to the concrete domain, which we discuss next.

Functionality of paths to the concrete domain As we mentioned, access to the concrete domain is given by paths to the elements on which we wish to place constraints. For paths of arbitrary length, only functional roles are allowed in the classical setting. This is an impactful restriction, since many useful real-world relationships are not functional, such as management or authorization roles, or sibling-type relationships. Non-functional roles are sometimes supported in the classical setting, usually by limiting their appearances to paths containing at most a single role [40, 41], which is a rather degenerate case. In contrast, $\mathcal{ALCF}^{\mathcal{P}}$ supports paths of arbitrary length composed of arbitrary roles, which significantly improve the modeling capabilities of the logic. We mention two notable exceptions in the classical landscape. The first one is $\mathcal{ALCFP}(\mathcal{D})$, which supports arbitrary paths of non-functional roles but disallows TBoxes [22]. The second is $\mathbb{Q}$-$\mathcal{SHIQ}$ [43], which supports pairing a *single* non-functional role and immediate access (an empty path) to the concrete domain, but does admit general TBoxes. However, the latter again involves an assumption – this time on the concrete domain – which $\mathcal{ALCF}^{\mathcal{P}}$ does not require, and this leads us to the final and perhaps most exciting reason to investigate $\mathcal{ALCF}^{\mathcal{P}}$.

Denseness of the concrete domain Concrete domains based on the integer or natural numbers have been singled out in the literature as very desirable extensions, since they allow apt modeling of various attributes, for example, the number of children a person has. A related feature which has been long sought-after is the ability to restrict the values an attribute takes to be integer or natural while generally operating over a dense domain such as the reals. However, despite their recognized importance, decidability results on extensions of $\mathcal{ALC}$ with non-dense domains remained elusive until recently, with the most general criterion for a concrete domain to preserve decidability being ω-admissibility, which inherently requires the domain to be dense. To our knowledge, there are but two works from which *mere decidability* results on $\mathcal{ALC}$ with integer domains can be inferred; the first being a result on CTL* with comparisons of integers [44], and the second being the work by Carapelle and Turhan on $\mathcal{ALCF}^{\mathcal{P}}$ [45].

The proofs in [44] and [45] do not suggest an elementary complexity bound, so the question of complexity is intriguing regardless of the technique used to answer it. We are interested in employing automata to answer this question in order to understand the cost of supporting non-dense domains, and whether we can leverage an automata-theoretic

answer to obtain results on related settings. In addition, considering that this has been a longstanding challenge, we hope to gain insight on what makes integer domains difficult to reason about in the context of DLs with concrete domains.

1.1.2 The logic $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$

The second logic we investigate fits into the line of research on extensions of FO^2 with built-in relations, which are assumed to be interpreted in the desired way without axiomatizing their properties. The aforementioned motivation of modeling data values from infinite (often numeric) domains guides the choice of a linear order and a preorder as the extending relations. A linear order would nicely model the sequential and one-dimensional nature of program traces, and a preorder in particular would be a good candidate for modeling data, as it is essentially an equivalence relation whose equivalence classes are linearly ordered. This allows the same data value to be assigned to multiple elements (as opposed to a linear order), while also allowing testing for both equality and comparisons (as opposed to an equivalence relation which only allows data value equality testing). In addition, we will include the successor relation of the preorder, and denote the logic $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$. The successor relation S_2 supports testing whether two data values are consecutive. As is typical, the addition of preorders can easily lead to undecidability, however, this logic has an ExpSpace-complete finite satisfiability problem. This was proved by Schwentick and Zeume [46] using a satisfiability-preserving reduction to a problem with a two-dimensional geometric flavor.

In this case, constructing automata for the logic would not answer a complexity question, but an automata model that maintains a parsimonious relationship between the logical models and the accepted inputs would allow one to employ the automaton not only for tasks which reduce to an emptiness test, but possibly also for tasks that reduce to a membership problem. In addition, there might be something to learn from exploring closure properties of such automata, especially negative ones.

1.2 Results

As we mentioned, decidability is already established for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, but no elementary complexity bounds are known. For this logic, we make the following contributions:

1. We provide an automata-theoretic upper bound for the satisfiability of concepts w.r.t. a general TBox in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ which matches the lower complexity bound of ordinary $\mathcal{ALC}$ and is thus optimal.

2. As part of the treatment, we needed to construct the product of Rabin tree automata, which has not been previously spelled out in the literature. We provide this construction, which is an easy adaptation of an unpublished proof by Yoad Lustig and Nir Piterman.

3. Our automata construction from the first item is altered to support dense concrete domains with predicates that ensure certain values are integer or natural, which have also been mentioned in the literature as a desired feature.

4. We translate related DLs with concrete domains into $\mathcal{ALCF}^{\mathcal{P}}$ and obtain new decidability and complexity results for the former, which are tight in some cases.

The first item is rather surprising, since it means that we can add integers to $\mathcal{ALCF}$ for free. Considering how decidability for similar DLs with concrete domains comes at a price, be it reflected in the concrete domain, in restrictions on the TBox, or in restrictions on the paths used, and considering how long this problem has been open despite being singled out as a very desirable extension – one would think that at the very least we would pay with higher computational complexity.

For the aforementioned $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$, there is already a tight complexity result but no corresponding automata model, which would provide a more intuitive understanding of the expressive power of the logic and allow us to reason about data words without explicit logical formulas. We make the following contributions:

1. We introduce the novel automata model Pebble-Intervals Automata (PIA), and show that we do not need the full data words in order to reason about them. Our PIA will operate on ordinary strings, which will be viewed as data words whose data values have been projected away. Furthermore, the PIA will only need to consider those elements of the data word that carry, in a sense, meaningful information.

2. We prove that the emptiness test for PIAs is PSpace in general, and NL-complete under some restrictions.

3. We show that PIAs subsume NFAs, accept some context-free and non context-free languages. If our suspicion that PIAs do not subsume CF languages is correct, this implies that they have a somewhat unusual relationship with the standard automata classes.

4. We show that PIAs are effectively closed under union, concatenation, Kleene-$\star$, shuffle and iterated shuffle, and are not effectively closed under intersection nor complement. As a consequence of the constructions in the non-closure proof, we will also have that their universality and inclusion problems are undecidable.

5. We provide an automata-theoretic proof of the upper complexity bound for the finite satisfiability problem of $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ by constructing an automaton for a given $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ sentence which maintains a *parsimonious* relationship between models and accepting computations.

6. Along the way, we also prove a normal form for $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ reminiscent of the Scott Normal Form.

Publications The content of the thesis is based on the following peer-reviewed publications:

- N. Labai, M. Homola, M. Ortiz, "Constructive Satisfiability Procedure for $\mathcal{ALC}^P(\mathcal{Z})$ (Preliminary Report)," *DL 2017*.
- N. Labai, M. Ortiz, M. Šimkus, "An ExpTime Upper Bound for $\mathcal{ALC}$ with Integers," *KR 2020*.
- N. Labai, T. Kotek, M. Ortiz, H. Veith, "Pebble-Intervals Automata and FO^2 with Two Orders," *LATA 2020*.

There have been two additional publications during the author's studies that are unrelated to the thesis:

- N. Labai, J.A. Makowsky, "On the Exact Learnability of Graph Parameters: The Case of Partition Functions," *MFCS 2016*.
- N. Labai, J.A. Makowsky, "Hankel Matrices for Weighted Visibly Pushdown Automata," *LATA 2016*.

1.3 Structure of the thesis

In Chapter 2 we provide basic definitions and notation which are used throughout the thesis, such as our handling of strings and trees, and complexity classes. Then, the thesis is divided into two parts.

In the first part, we investigate the incorporation of integer domains into DLs. We begin by providing preliminaries on DLs in Chapter 3, which include, in addition to the usual syntax and semantics, background on and construction of Rabin tree automata. Chapter 4 gives a comprehensive introduction to the DL $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ as well as a normal form result. In the following Chapter 5 we do the heavy lifting of Part I by setting up theoretical groundwork and then constructing Rabin tree automata for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$. We then adapt our theoretical groundwork and construction to a real numbers setting in Chapter 6, and allow the presence of a predicate for enforcing integer values while maintaining complexity bounds. Chapter 7 rounds out this part, where we compare $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ to other DLs with concrete domains, and in some cases, provide translations from them into $\mathcal{ALCF}^{\mathcal{P}}$ which deliver new complexity results.

In the second part, we study the automata-logic connection for $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$. In Chapter 8, we introduce the novel Pebble-Intervals Automata (PIA) model, and conduct their usual investigation; we demonstrate their computational power, show they have a decidable emptiness test, study their closure properties, and present some non-closure properties. We pivot in Chapter 9 to discuss the two-variable fragment of First Order logic and extensions thereof that facilitate data modeling. We present the logic $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$

and the notion of data words, and finish the chapter by proving a normal form for $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$. In Chapter 10 we tie the two previous chapters together and present the main result of this part, which is that PIAs encompass string projection languages of $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ sentences. For this, we develop several notions to overcome the inherent gap between PIA having finite-memory and data words having no bound on their data values. Culminating this development is our construction of a PIA for a given $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ sentence, which provides an automata-theoretic proof for the upper bound of the finite satisfiability problem of $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$.

Finally, in Chapter 11 we discuss our results and provide outlook for further research.

CHAPTER 2

Basic Definitions and Notation

In this chapter, we provide some general definitions and notation which will be used throughout the thesis.

We denote the set of integers by $\mathbb{Z}$, the set of natural numbers by $\mathbb{N}$ and by $\mathbb{N}^+$ the set of strictly positive natural numbers, the set of reals by $\mathbb{R}$, and the set of rational numbers by $\mathbb{Q}$.

For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, n\}$. Note that $[0] = \emptyset$.

All alphabets in the thesis are assumed to be finite unless stated otherwise.

Strings A *(finite) string* of length $n \in \mathbb{N}$ over alphabet Σ is a mapping $w : [n] \to \Sigma$, written also $w = w(1) \cdots w(n)$, where for all $\ell \in [n]$, $w(\ell) = \sigma$ for $\sigma \in \Sigma$. Note that $w : [0] \to \Sigma$ is the empty string ε. We denote the length of w by $|w|$. An *infinite string* is a mapping $\mathbf{w} : \mathbb{N}^+ \to \Sigma$, and will be printed in boldface.

Trees We introduce some notation for our treatment of trees. Recall that a graph G consists of its vertex set $V(G)$ and its edge set $E(G)$, and a tree is a connected acyclic graph. We will normally treat trees in the following way, which is reminiscent of the Trie data structure (see e.g. [47]):

Definition 2.1 (n-tree)**.** *For $n \in \mathbb{N}^+$, an n-tree T is given by a set* $\mathrm{dom}(T) \subseteq [n]^\star$ *of* nodes*, which may be infinite, that is closed under prefixes. That is, if $u \in \mathrm{dom}(T)$ then also every prefix of u is a node of T.*

For $i \in [n]$ and $u \in [n]^\star$ where $ui \in \mathrm{dom}(T)$, we say that u is the parent of ui, and for $uv \in \mathrm{dom}(T)$ where $|v| = j$ we say that u is the j-ancestor of uv. When $\mathrm{dom}(T) = [n]^\star$ we say that T is a full n-tree.

We assume all trees are full unless stated otherwise, and we continue our presentation under this assumption. All definitions can be easily adapted.

Trees will often carry a letter from some alphabet Σ on their nodes. In this case we treat T as a mapping $[n]^\star \to \Sigma$ and refer to T as a tree over Σ. We denote by $T(v)$ the letter on the node v.

In the first part of the thesis, we will also sometimes restrict our attention to subtrees. Following [48]:

Definition 2.2 (Subtree)**.** *For an n-tree T over Σ, the* subtree rooted at $w \in [n]^\star$ *is the tree $T|_w(v) = T(wv)$ for all $v \in [n]^\star$.*

Definition 2.3 (Regular trees)**.** *We say that an n-tree T over Σ is* regular *if the set $\{T|_u \mid u \in [n]^\star\}$ of subtrees of T is finite.*

Finally, we discuss the concept of tree decompositions. Loosely speaking, the tree decomposition of a graph G is a division of G into bags which, when connected according to the edges in G, induce a tree-like structure.

Definition 2.4 (Tree decomposition)**.** *Let G be a graph, let T be a tree, and let $\mathcal{V} = (V_t)_{t \in T}$ be a family of vertex sets $V_t \subseteq V(G)$ we call bags, indexed by the vertices of T. We say that $(T, \mathcal{V})$ is a tree decomposition of G if*

1. *$V(G) = \bigcup_{t \in T} V_t$, that is, every vertex of G belongs to at least one bag,*
2. *for every edge e of G, there exists some $t \in T$ such that both endpoint of e are in V_t, that is, all edges of G are recoverable from $(T, \mathcal{V})$, and finally,*
3. *for $t_1, t_2, t_3 \in T$, if t_3 is on the (unique) path between t_1 and t_2, then $t_1 \cap t_2 \subseteq t_3$. In other words, the bags in which a vertex of G appears form a connected component.*

There may be several tree decompositions for a given graph, and typically one is interested in the tree decompositions which minimize the size of the bags used. In our encounters with tree decompositions, we will also be interested in keeping the size of the bags small, but it will not be the most important thing. Figure 2.1 presents a graph whose structure closely resembles a tree already (and is typical of the graphs we will see in later chapters) and a tree decomposition for it. For more details on tree decompositions, we refer to [49].

Complexity classes We assume the reader is familiar with basic computability and complexity theory, and refer to [50, 51] for a comprehensive treatment. We briefly recall some complexity classes encountered in the thesis.

- NL – the set of problems that can be decided with a non-deterministic Turing machine using logarithmic space.

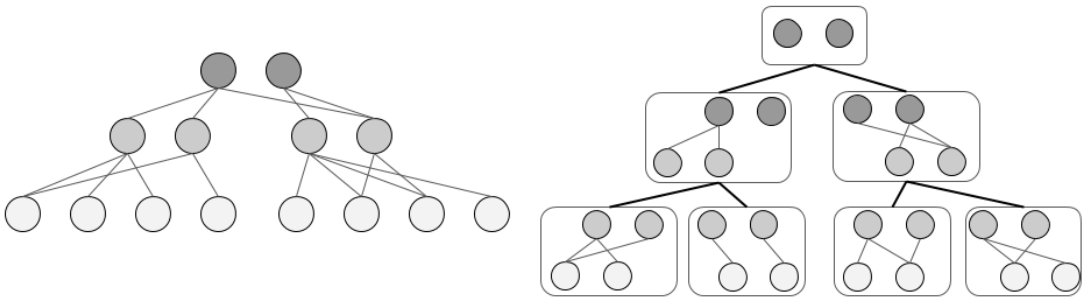


Figure 2.1: On the left, a graph that is already tree-like, and on the right, its tree decomposition. The vertices are only colored to clarify which bag contains which vertices.

- PSPACE – the set of problems that can be decided with a deterministic Turing machine using polynomial space. Keep in mind that by Savitch's Theorem [52], we have PSPACE = NPSPACE where NPSPACE is the non-deterministic version of PSPACE.
- EXPSPACE – the set of problems that can be decided with a deterministic Turing machine using exponential space. Again by Savitch's Theorem we have EXPSPACE = NEXPSPACE.
- EXPTIME – the set of problems that can be decided with a deterministic Turing machine using exponential time.

Part I

Data Values in Description Logics

CHAPTER 3

Preliminaries on Description Logics

As mentioned before, Description Logics use concepts and roles to model domain knowledge. The syntax of a description logic is typically defined inductively, with some set of atomic concept names and a set of role names, and constructors for building more complex concepts and roles. In this chapter, we present the syntax and semantics of the 'basic' description logic $\mathcal{ALC}$, which does not actually allow the construction of complex roles. Its concepts are constructed in a way that is similar to how formulas in First Order logic are constructed, with boolean combinations and $\mathcal{ALC}$'s versions of existential and universal quantification. We also present its extension with functional roles, denoted $\mathcal{ALCF}$. We review well-known results on the reasoning task of our interest, namely concept satisfiability w.r.t. TBoxes. We refer to [53] for more on Description Logics.

Decidable logics often have the tree model property, which basically means that any satisfiable formula has some tree-shaped model. Many description logics have this property, which is one of the reasons tree automata are so commonly used for proving decidability and complexity bounds. We will also be using tree automata in this thesis, namely, Rabin tree automata [54]. We discuss their properties and spell out a polynomial-time construction of their product (Lemma 3.13).

3.1 $\mathcal{ALC}$

Many consider the DL $\mathcal{ALC}$, introduced by Schmidt-Schauß and Smolka [55], to be the baseline DL to which other DLs are compared. The set of $\mathcal{ALC}$ concepts is defined inductively.

Definition 3.1 ($\mathcal{ALC}$ concepts)**.** *Let* $\mathsf{N_C}$ *be a countable infinite set of* concept names *and let* $\mathsf{N_R}$ *be a countable infinite set of* role names*. The set of* $\mathcal{ALC}$ *concepts is the smallest set where*

- *every concept name* $A \in \mathsf{N_C}$ *is a concept*
- *if* C *and* D *are concepts and* $r \in \mathsf{N_R}$ *is a role name, then the following are* $\mathcal{ALC}$ *concepts:*

$$\neg C, \quad C \sqcap D, \quad C \sqcup D, \quad \exists r.C, \quad \forall r.C$$

We use $\top$ *as shorthand for* $A \sqcup \neg A$ *and* $\bot$ *as shorthand for* $\neg\top$*.*

Example 3.2. *With* Koala*,* Australia*, and* Eucalyptus *as concept names, and* eats *and* nativeTo *as role names, we can build the* $\mathcal{ALC}$ *concept*

$$\mathsf{Koala} \sqcup \neg\exists \mathsf{nativeTo}.\mathsf{Australia}$$

Next we define TBoxes, which contain the axioms about the world.

Definition 3.3 (TBox axiom, general TBox)**.** *A* TBox axiom *has the form* $C \sqsubseteq D$*, where* C *and* D *are concepts. Axioms are also sometimes called General Concept Inclusions (GCI). A (general)* TBox $\mathcal{T}$ *is a finite set of TBox axioms.*

We use $C \equiv D$ as shorthand for the two axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

Example 3.4. *We can assert that koalas are herbivores and only eat eucalyptus leaves, and that herbivores do not eat meat, as follows:*

$$\begin{aligned} \mathsf{Koala} &\sqsubseteq \mathsf{Herbivore} \\ \mathsf{Koala} &\sqsubseteq \forall \mathsf{eats}.\mathsf{EucalyptusLeaves} \\ \mathsf{Herbivore} \sqcap \exists \mathsf{eats}.\mathsf{Meat} &\equiv \bot \end{aligned}$$

In order to define the semantics of $\mathcal{ALC}$, we first need to define interpretations.

Definition 3.5 ($\mathcal{ALC}$-Interpretation)**.** *An* $\mathcal{ALC}$-interpretation *is a pair* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *where* $\Delta^{\mathcal{I}}$ *is a non-empty set called* the domain *and* $\cdot^{\mathcal{I}}$ *is 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* $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*.*

Definition 3.6 (Semantics of $\mathcal{ALC}$)**.** *The semantics of $\mathcal{ALC}$ are given by inductively extending the interpretation function $\cdot^{\mathcal{I}}$ of an $\mathcal{ALC}$-interpretation $\mathcal{I}$ as follows:*

$$\begin{aligned}
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C_1 \sqcap C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(C_1 \sqcup C_2)^{\mathcal{I}} &= C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}} \\
(\exists r.C)^{\mathcal{I}} &= \{u \mid \exists v \in C^{\mathcal{I}} \text{ such that } (u,v) \in r^{\mathcal{I}}\} \\
(\forall r.C)^{\mathcal{I}} &= \{u \mid \forall v \in \Delta^{\mathcal{I}}, \text{ if } (u,v) \in r^{\mathcal{I}} \text{ then } v \in C^{\mathcal{I}}\}
\end{aligned}$$

Given a concept and/or a TBox the natural question is whether they are satisfiable.

Definition 3.7 (Concept satisfiability w.r.t. a TBox and pure concept satisfiability)**.** *An interpretation $\mathcal{I}$ is a* model *of a TBox $\mathcal{T}$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all $C \sqsubseteq D \in \mathcal{T}$.*

Given a concept C and TBox $\mathcal{T}$, we say that C is satisfiable w.r.t. *$\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. In this case we may denote $\mathcal{I} \models_{\mathcal{T}} C$.*

Pure concept satisfiability *is the problem of determining whether C is satisfiable w.r.t. an empty TBox, i.e. if there is an interpretation $\mathcal{I}$ for which $C^{\mathcal{I}} \neq \emptyset$.*

Theorem 3.8 ([56] and [57], and an alternative proof in [58])**.** *Pure concept satisfiability for $\mathcal{ALC}$ is PSpace-complete, and concept satisfiability w.r.t. a general TBox is ExpTime-complete.*

There are numerous extensions of $\mathcal{ALC}$ with constructors that allow one to express that e.g. certain roles are functional, use inverses of roles, compose roles, etc. The newly added capability is conventionally reflected by appending a letter to $\mathcal{ALC}$, for example, $\mathcal{ALCI}$ for inverses.

In this thesis, we are interested in $\mathcal{ALCF}$, which is obtained from $\mathcal{ALC}$ by supporting *functional roles*. We discuss $\mathcal{ALCF}$ next.

3.2 $\mathcal{ALCF}$

$\mathcal{ALCF}$ is the extension of $\mathcal{ALC}$ with functionality, meaning that some roles are required to be functional. Precisely speaking, we let $\mathsf{N_F} \subseteq \mathsf{N_R}$ be a countable infinite set of *functional role names* such that $\mathsf{N_R} \setminus \mathsf{N_F}$ is also infinite, and we require that for an $\mathcal{ALCF}$-interpretation $\mathcal{I}$, in addition to the conditions in Definition 3.5, $\mathcal{I}$ satisfies that if $r \in \mathsf{N_F}$ then $r^{\mathcal{I}}$ is functional, i.e. if both $(u,v) \in r^{\mathcal{I}}$ and $(v,u) \in r^{\mathcal{I}}$ then $u = v$. An example of a naturally functional role would be the place of birth.

As with $\mathcal{ALC}$, pure concept satisfiability in $\mathcal{ALCF}$ is PSpace-complete [59]. ExpTime-completeness of concept satisfiability w.r.t. general TBoxes can be inferred from [60, 61], since $\mathcal{ALCF}$ is a special case. We will provide an explicit automata theoretic proof later in the chapter, as well.

3.3 Rabin tree automata

A popular way to solve satisfiability for DLs is to reduce it to the (non-)emptiness of automata. Typically, one shows that if there is a model of a concept C w.r.t. a TBox $\mathcal{T}$, then there is an automaton with a non-empty language. Of course, there are some restrictions – we need to be able to effectively construct said automaton, and the automata model needs to have a decidable emptiness problem. In order to achieve this, one often exploits a tree model property of the DL, which essentially states that if there is a model, then there is a *tree model*, i.e. if we ignore the role names connecting the logical elements, the underlying graph is a tree. A logic having the tree model property has been identified as an encouraging sign for it having a decidable satisfiability problem, also outside the scope of DLs [62].

There is a variety of tree automata with decidable emptiness problems, and the choice of variant depends on the expressive properties of the DL at hand. We refer to [48] for a survey on automata on infinite trees, and to [53] for demonstrations of this technique.

In this thesis, we will be constructing Rabin tree automata for DLs that will run on infinite trees.

Definition 3.9 (Rabin tree automaton)**.** *A* Rabin tree automaton *over the alphabet* Σ *has the form* $\mathcal{A} = (Q, q_0, \delta, \Omega)$ *with*

1. *a finite state set* Q*,*
2. *initial state* q_0*,*
3. *transition relation* $\delta \subseteq Q \times \Sigma \times Q^n$*, and*
4. $\Omega = \{(L_1, U_1), \ldots, (L_m, U_m)\}$ *is a collection of state sets* $L_i, U_i \subseteq Q$ *which we refer to as accepting pairs.*

We also denote $(q, \sigma) \ni (q_1, \ldots, q_n)$ *for* $(q, \sigma, q_1, \ldots, q_n) \in \delta$*.*

For a Rabin tree automaton $\mathcal{A}$, we sometimes denote its state set by $Q(\mathcal{A})$, its set of accepting pairs by $\Omega(\mathcal{A})$ and its language by $\mathscr{L}(\mathcal{A})$. For technical reasons that are irrelevant to our treatment, we assume there is indeed a *single* initial state, which is only visited at the root of the input tree.

Let T be an (infinite) n-tree over Σ. That is, its set of nodes is $[n]^\star$ and T is essentially a mapping from $[n]^\star$ to Σ. A run of a Rabin tree automaton $\mathcal{A}$ can be thought of as annotating T with the state $\mathcal{A}$ is in when it visits each node and reads the letter at the node. More precisely:

Definition 3.10 (Semantics of Rabin tree automata)**.** *A* run *of* $\mathcal{A}$ *on an infinite tree* T *is a mapping* $\rho : [n]^\star \to Q$ *of the nodes of* T *into states which respects the transition relation. That is, with*

- $\rho(\varepsilon) = q_0$ *and*
- $(\rho(w), T(w), \rho(w1), \ldots, \rho(wn)) \in \delta$ *for* $w \in [n]^\star$.

For a path π *in* T *and a run* ρ *denote by* $\mathrm{In}(\rho \mid \pi)$ *the set of states that appear infinitely often in the restriction of* ρ *to* π. *A run* ρ *of* $\mathcal{A}$ *is* successful *if*

$$\begin{gathered}\text{for all paths } \pi \text{ there exists an } i \in [m] \text{ with} \\ \mathrm{In}(\rho \mid \pi) \cap L_i = \emptyset \text{ and } \mathrm{In}(\rho \mid \pi) \cap U_i \neq \emptyset.\end{gathered}$$

A tree T *is* accepted *by the Rabin tree automaton if some run of* $\mathcal{A}$ *on* T *is successful.*

That is, there only needs to be one successful run on T to accept it, but in order for a run to be successful, every path in T has to satisfy the acceptance condition. We state some properties of Rabin tree automata which will be useful later.

Theorem 3.11 (Emptiness [63])**.** *Emptiness of a Rabin tree automaton* $\mathcal{A}$ *is decidable in time polynomial in* $|Q(\mathcal{A})|$ *and exponential in* $|\Omega(\mathcal{A})|$.

By inspecting the treatment of Muller and Schupp of alternating tree automata in [64], we can state:

Theorem 3.12 (Complement of Rabin tree automaton with a constant number of accepting pairs)**.** *Given a Rabin tree automaton* $\mathcal{A}$ *with a constant number of accepting pairs, one can construct a Rabin tree automaton* $\mathcal{A}^c$ *such that* $\mathscr{L}(\mathcal{A}^c)$ *is the complement of* $\mathscr{L}(\mathcal{A})$, *where* $|Q(\mathcal{A}^c)|$ *is exponential in* $|Q(\mathcal{A})|$ *and* $|\Omega(\mathcal{A}^c)|$ *is polynomial in* $|Q(\mathcal{A})|$.

We provide a construction of the product of two Rabin tree automata, which is an easy adaptation of a privately communicated proof by Yoad Lustig and Nir Piterman for Streett automata.

Lemma 3.13 (Product)**.** *Given Rabin tree automata* $\mathcal{A}$ *and* $\mathcal{A}'$, *one can construct a Rabin tree automaton* $\mathcal{A}^\cap$ *such that* $\mathscr{L}(\mathcal{A}^\cap) = \mathscr{L}(\mathcal{A}) \cap \mathscr{L}(\mathcal{A}')$, *where* $|Q(\mathcal{A}^\cap)| = |Q(\mathcal{A})| \cdot |Q(\mathcal{A}')|$ *and* $|\Omega(\mathcal{A}^\cap)| = |\Omega(\mathcal{A})| \cdot |\Omega(\mathcal{A}')|$.

Proof. The product of two Rabin tree automata is obtained by simply taking the product of the state sets, transition relation, and accepting pairs as follows. Let

$$\mathcal{A} = (Q, q_0, \delta, \{(L_1, U_1), \ldots, (L_m, U_m)\}), \quad \mathcal{A}' = (Q', q_0', \delta', \{(L_1', U_1'), \ldots, (L_{m'}', U_{m'}')\})$$

be two Rabin tree automata over some alphabet Γ which run on n-trees. The automaton $\mathcal{A}^\cap = (Q^\cap, q_0^\cap, \delta^\cap, \Omega^\cap)$ is given by

- $Q^\cap = Q \times Q'$

- $q_0^{\cap} = (q_0, q_0')$
- $$((q, q'), \gamma, (q_1, q_1'), \ldots, (q_n, q_n')) \in \delta^{\cap}$$
  if and only if
  $$(q, \gamma, q_1, \ldots, q_n) \in \delta \quad \text{and} \quad (q', \gamma, q_1', \ldots, q_n') \in \delta'$$
- $\Omega^{\cap} = \{((L_i, L_j'), (U_i, U_j')) \mid i \in [m], j \in [m']\}$

That is, $\mathcal{A}^{\cap}$ runs $\mathcal{A}$ and $\mathcal{A}'$ simultaneously. For a run on some input, we have acceptance by both $\mathcal{A}$ and $\mathcal{A}'$ if and only if we have that every path has some i such that its restriction to Q is successful due to (L_i, U_i), and some j such that its restriction to Q' is successful due to (L_j', U_j'). Therefore, a given input is accepted by both $\mathcal{A}$ and $\mathcal{A}'$ if and only if there is a run where every path has some (i, j) witnessing its success, i.e. there is an accepting run of $\mathcal{A}^{\cap}$. □

Finally, we mention the following celebrated theorem by Rabin, which can be found in various phrasing in the literature:

Theorem 3.14 (Rabin's Theorem ([65]))**.** *Any non-empty Rabin recognizable set of trees contains a regular tree.*

3.4 Rabin tree automata for $\mathcal{ALCF}$

Here we construct a Rabin tree automaton for checking the satisfiability of $\mathcal{ALCF}$ concepts w.r.t. general TBoxes. Our construction is an easy adaptation of a well known construction of a looping automaton which accepts exactly the tree models of an $\mathcal{ALC}$ concept w.r.t. a TBox, see e.g. [66]. Our adaptation merely ensures functionality of the roles in $\mathsf{N_F}$.

Given an $\mathcal{ALCF}$ concept $\mathcal{C}$ and TBox $\mathcal{T}$, this automaton runs on trees over an alphabet Ξ which consists of sets of the concept names in $\mathcal{C}$ and $\mathcal{T}$ and a single role name from $\mathcal{C}, \mathcal{T}$. Following the notation in [66], let $S_{\mathcal{C},\mathcal{T}}$ be the set of subexpressions of $\mathcal{C}$ and $\mathcal{T}$, and let $R_{\mathcal{C},\mathcal{T}}$ be the set of role names used in $\mathcal{C}$ and $\mathcal{T}$. The role name in each letter indicates the role with which a logical element is connected to its parent. The states of this automaton are maximal consistent sets of the subexpressions in $\mathcal{C}, \mathcal{T}$, also known as Hintikka sets:

Definition 3.15 (Hintikka set)**.** *The* Hintikka sets *for $\mathcal{C}, \mathcal{T}$ are the subsets $q \subseteq S_{\mathcal{C},\mathcal{T}} \cup R_{\mathcal{C},\mathcal{T}}$ such that either $q = \emptyset$, or q satisfies all the following:*

- *q contains exactly one role name,*

- *if* $\top \sqsubseteq D \in \mathcal{T}$ *then* $D \in q$,
- *if* $C_1 \sqcap C_2 \in q$ *then* $\{C_1, C_2\} \subseteq q$,
- *if* $C_1 \sqcup C_2 \in q$ *then* $\{C_1, C_2\} \cap q \neq \emptyset$,
- $\{A, \neg A\} \not\subseteq q$ *for every concept name* A.

The construction is very nearly identical to the one in [66], therefore we only describe the parts needed to understand our adaptation. Note that given the number of existential restrictions occurring in $\mathcal{C}$ and $\mathcal{T}$, we can determine some n such that if $\mathcal{C}$ is satisfiable w.r.t. $\mathcal{T}$, then it has a (full) tree model of degree n.

The state set of $\mathcal{A}_{\text{alcf}}$ contains the Hintikka sets for $\mathcal{C}, \mathcal{T}$, i.e. $q \subseteq S_{\mathcal{C},\mathcal{T}} \cup R_{\mathcal{C},\mathcal{T}}$ where either $q = \emptyset$ or q is a maximally consistent set which contains, in addition to subexpressions from $S_{\mathcal{C},\mathcal{T}}$, exactly one role name.

Our automaton $\mathcal{A}_{\text{alcf}} = (Q, q_0, \delta_{\text{alcf}}, (\emptyset, Q))$ has its transition relation only differ from the one in [66] in order to properly handle functional roles. Specifically, our $(q, \xi) \ni (q_1, \ldots, q_n)$ additionally satisfies that

- $Q = \{q \mid q \text{ is a Hintikka set of } \mathcal{C}, \mathcal{T}\}$
- We have $(q, \xi) \ni (q_1, \ldots, q_n)$ satisfying
  - q and ξ have the same concept and role names contained in them;
  - if $q = \emptyset$ then $q_i = \emptyset$ for every $i \in [n]$;
  - if $\exists r.D \in q$, then there is an i such that $\{D, r\} \subseteq q_i$, furthermore, if $r \in \mathsf{N_F}$, then there is exactly one such i;
  - if $\forall r.D \in q$ and $r \in q_i$, then $D \in q_i$, furthermore, if $r \in \mathsf{N_F}$, then there is at most one $i \in [n]$ such that $r \in q_i$.

Proposition 3.16. *$\mathcal{A}_{\text{alcf}}$ accepts exactly the tree models of $\mathcal{C}$ w.r.t. $\mathcal{T}$.*

Observation 3.17. *The alphabet Ξ and the number of states of $\mathcal{A}_{\text{alcf}}$ are exponential in the size of $\mathcal{C}$ and $\mathcal{T}$, and the Ω of $\mathcal{A}_{\text{alcf}}$ has one pair.*

CHAPTER 4

The Description Logic $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_C)$

Here we enrich $\mathcal{ALCF}$ with the ability to apply constraints on integer values, following the treatment of Carapelle and Turhan [38]. There, the logic $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{D})$ was defined for arbitrary domains $\mathcal{D}$, however, we restrict our attention to $\mathcal{Z}_c$ and adapt the definitions to that setting. The most general results on DLs with concrete domains are for extensions of $\mathcal{ALC}(\mathcal{C})$ with *ω-admissible domains*. There, concept satisfiability w.r.t. general TBoxes remains decidable [42], but there are two key restrictions on that setting which are relaxed here: first is that the concrete domain is dense, and the second is that only functional roles occur in the paths connecting to the concrete domain. Already in the seminal work of Baader and Hanschke [20] they point out the potential usefulness of allowing referral to the concrete domains also along paths of regular roles, but this easily results in undecidability. For example, such an extension of $\mathcal{ALC}$ known as $\mathcal{ALCFP}(\mathcal{D})$ is undecidable for any so-called *arithmetic domain* $\mathcal{D}$ [22]. However, $\mathcal{Z}_c$ and its analogue over the real numbers $\mathcal{R}_c$ are not arithmetic, and the corresponding DLs $\mathcal{ALCFP}(\mathcal{Z}_c)$ and $\mathcal{ALCFP}(\mathcal{R}_c)$ do not seem to have been studied before. We will later encode these logics into $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ and $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_c)$ and prove that their satisfiability problem is decidable and obtain upper complexity bounds (which are tight under some restrictions).

But before that, in this chapter we present the syntax and semantics of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ and demonstrate its expressive power with several examples. Despite the difference in notation, we will sometimes want to make the distinction between $\mathcal{ALCF}$ and $\mathcal{ALCF}^{\mathcal{P}}$ abundantly clear. In these cases, we will refer to $\mathcal{ALCF}$ concepts as *plain* concepts, and similarly for TBoxes and interpretations. We finish the chapter by proving the existence of a normal form (Lemma 4.12) which will facilitate the aforementioned developments in the next chapter.

4.1 Syntax and semantics of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$

The domain $\mathcal{Z}_c$ is comprised of the integers $\mathbb{Z}$, the binary relations $<$ and $=$, and unary predicates $=c$ for equality with integer constants $c \in \mathbb{Z}$.

Here we provide syntax and semantics for the case $\mathcal{Z}_c = \langle \mathbb{Z}, <, =, \{= c \mid c \in \mathbb{Z}\} \rangle$. First, we define the constraints over $\mathcal{Z}_c$ allowed in $\mathcal{ALCF}^{P}(\mathcal{Z}_c)$.

Let Reg be a countably infinite set of register names.

Definition 4.1 (Register term)**.** *A* register term *is an expression of the form $S^k x$, where $x \in \mathsf{Reg}$ and $k \geq 0$ is an integer.*

Definition 4.2 (Constraint)**.** *An* atomic constraint *is an expression of the form*

$$t = t', \quad t < t', \text{ or } \quad t = c$$

where t, t' are register terms, and $c \in \mathbb{Z}$. A (complex) constraint Θ *is an expression built from atomic constraints using the Boolean connectives $\neg, \wedge$ and $\vee$. The* depth *of Θ (in symbols, depth(Θ)) is the maximal d such that some register term $S^d x$ appears in Θ.*

Example 4.3. *Some simple examples of constraints would be*

$runningRecord < previousRunningRecord$

$carDrivingAge = 18$

$\neg(airPressure < seaLevelAirPressure) \vee (waterBoilingTmp < standardWaterBoilingTmp)$

Definition 4.4 (Path constraint)**.** *A* role path *P is any finite sequence $r_1 \cdots r_n$ of role names, with $n \geq 0$. We denote the length of P by $|P|$, i.e. $|P| = n$. The empty sequence is also a role path, which we denote with ε. Expressions of the form $D = \exists P.[\![\Theta]\!]$ and $D = \forall P.[\![\Theta]\!]$ where Θ is a constraint of depth at most n are called* path constraints*. The depth of a path constraint is the length of the path, i.e. depth$(D) = |P|$.*

Definition 4.5 ($\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concepts)**.** *The set of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concepts is the minimal augmentation of $\mathcal{ALCF}$ concepts such that every path constraint is an $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concept.*

Definition 4.6 ($\mathcal{Z}_c$-interpretation)**.** *A $\mathcal{Z}_c$-*interpretation *is a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \beta)$, consisting of a non-empty set $\Delta^{\mathcal{I}}$ (called* domain*), a* register function *$\beta : \Delta^{\mathcal{I}} \times \mathsf{Reg} \to \mathbb{Z}$, and a* (plain) interpretation function *$\cdot^{\mathcal{I}}$ 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 $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Furthermore, if $r \in \mathsf{N_F}$ then $r^{\mathcal{I}}$ is functional, i.e. if both $(u, v) \in r^{\mathcal{I}}$ and $(v, u) \in r^{\mathcal{I}}$ then $u = v$.*

Definition 4.7 (Semantics of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$)**.** *Let $\mathcal{I}$ be a $\mathcal{Z}_c$-interpretation. For a tuple $\vec{v} = (v_0, \ldots, v_n)$ of elements of $\Delta^{\mathcal{I}}$, and constraints Θ_1, Θ_2 we define the following:*

- *$\mathcal{I}, \vec{v} \models (S^i x = S^j y)$ if and only if $\beta(v_i, x) = \beta(v_j, y)$,*

- $\mathcal{I}, \vec{v} \models (S^i x < S^j y)$ *if and only if* $\beta(v_i, x) < \beta(v_j, y)$,
- $\mathcal{I}, \vec{v} \models (S^i x = c)$ *if and only if* $\beta(v_i, x) = c$;
- $\mathcal{I}, \vec{v} \models \Theta_1 \wedge \Theta_2$ *if and only if* $\mathcal{I}, \vec{v} \models \Theta_1$ *and* $\mathcal{I}, \vec{v} \models \Theta_2$;
- $\mathcal{I}, \vec{v} \models \Theta_1 \vee \Theta_2$ *if and only if* $\mathcal{I}, \vec{v} \models \Theta_1$ *or* $\mathcal{I}, \vec{v} \models \Theta_2$;
- $\mathcal{I}, \vec{v} \models \neg\Theta_1$ *if and only if* $\mathcal{I}, \vec{v} \not\models \Theta_1$.

For a role path $P = r_1 \cdots r_n$, *we define*

$$P^{\mathcal{I}} = \{(v_0, \ldots, v_n) \in \Delta^{n+1} \mid (v_0, v_1) \in r_1^{\mathcal{I}}, \ldots, (v_{n-1}, v_n) \in r_n^{\mathcal{I}}\}$$

The function $\cdot^{\mathcal{I}}$ *is extended to path constraints as follows:*

$$\begin{aligned}(\exists P.[\![\Theta]\!])^{\mathcal{I}} &= \{u \mid (u, \vec{v}) \in P^{\mathcal{I}} \text{ and } \mathcal{I}, (u, \vec{v}) \models \Theta\} \\ (\forall P.[\![\Theta]\!])^{\mathcal{I}} &= \{u \mid \text{for every } u \in \Delta^{\mathcal{I}}, \text{ if } (u, \vec{v}) \in P^{\mathcal{I}} \text{ then } \mathcal{I}, (u, \vec{v}) \models \Theta\}\end{aligned}$$

The function $\cdot^{\mathcal{I}}$ *of* $\mathcal{I}$ *extends to other complex concepts as in Definition 3.6. A* $\mathcal{Z}_c$*-interpretation* $\mathcal{I}$ *is a* model *of a TBox* $\mathcal{T}$, *if* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ *for all* $C \sqsubseteq D \in \mathcal{T}$.

Example 4.8. *The next axiom identifies* active departments *whose employees lead projects started in the last two years:*

$$\exists \mathsf{employs}\, \mathsf{leadsProject}[\![S^2 year > 2018]\!] \sqsubseteq \mathsf{ActiveDept}$$

In general, the roles '$\mathsf{employs}$*' and '*$\mathsf{leadsProject}$*' need not be functional.*

This simple examples demonstrates that $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ does not enjoy the finite model property:

Example 4.9 (No finite model property)**.** *The TBox with the axiom*

$$\top \sqsubseteq \exists r.[\![S^0 x < S^1 x]\!]$$

enforces an infinite chain of objects whose x *registers store increasing integer values.*

Just as with Definition 3.7, given a concept C and a Tbox $\mathcal{T}$ in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, we say that C is satisfiable w.r.t. $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.

$\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ enjoys a special case of the tree model property [45], where the tree models are the full tress described in the next definition:

Definition 4.10 (Tree-shaped interpretation)**.** *We say* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *is* tree-shaped *if* $\Delta^{\mathcal{I}} = \Gamma^\star$ *for a finite alphabet* Γ, *and for every* $u, v \in \Delta^{\mathcal{I}}$, *we have that* $(u, v) \in r^{\mathcal{I}}$ *for some* $r \in \mathsf{N_R}$ *if and only if* $v = ud$ *for some* $d \in \Gamma$. *Let* $d_1, \ldots d_k \in \Gamma$. *If* $ud_k \in \Delta^{\mathcal{I}}$, *we call* u *the* parent *of* ud_k, *and if* $v = ud_1 \cdots d_k \in \Delta^{\mathcal{I}}$, *we call* u *the* k-th ancestor *of* v. *We say* $\mathcal{I}$ *is an* n-tree interpretation *if* Γ *has size* n.

4.1.1 Previous Decidability Results of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{D})$

Already in [45], Carapelle and Turhan showed that $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{D})$ concept satisfiability w.r.t. general TBoxes is decidable for any negation-closed domain $\mathcal{D}$ with the so-called EHD property, which stands for "Existence of Homomorphism is Definable". This property holds when one can express the existence of a homomorphism from a structure to $\mathcal{D}$ in the logic BMWB, see [67, 68]. By using the tree model property of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{D})$, they replace path constraints with new concepts marking nodes in the tree model which satisfy the constraint. Then, they add relations that group nodes which – together – satisfy a path-constraint. This results in two structures; one is a (plain) tree-shaped $\mathcal{ALCF}$ interpretation and a corresponding tree-like structure with some additional information, that is not unlike the graph in the left hand side of Figure 2.1. This tree-like structure is called a *constraint graph*, and what remains is to map the constraint graph into the concrete domain. If such a mapping exists, then there are values we can assign to the registers that would satisfy the original constraints.

All of the above can be translated into BMWB, which is decidable over trees [68]. However, this proof of decidability uses several reductions to bespoke automata models, which unfortunately loses the ability to infer elementary complexity bounds on $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{D})$.

What we will be doing in the remainder of our treatment is providing a straightforward way of checking whether constraint graphs obtained from $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concept and TBox pairs are embeddable into $\mathbb{Z}$. Thanks to the mathematical justifications given in [45] and the references therein, we can focus on the structural properties of constraint graphs that ensure such embeddings exist. This is as opposed to providing a full argument, so to speak, that embeddability indeed implies satisfiability by e.g. constructing a satisfying interpretation from an embeddable constraint graph. Such arguments are conveniently provided in the aforementioned works.

4.2 Atomic Normal Form

Although we will not need to provide explicit constructions of satisfying interpretations in the later portions of our treatment, here we still need to do some technical work by providing a normal form for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concepts, which will nonetheless require us to explicitly describe interpretations. This normal form will allow us to only consider path constraints of length 1 containing only atomic constraints, which will be much easier to construct automata for later on.

Definition 4.11 (Atomic normal form)**.** *An $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$-concept is in* atomic normal form (ANF) *if for every $\exists P.[\![\Theta]\!]$ and $\forall P.[\![\Theta]\!]$ that appears in it, Θ is an atomic constraint and $|P| \leq 1$. A TBox $\mathcal{T}$ is in ANF if the TBox-concept $\bigsqcap_{C \sqsubseteq D \in \mathcal{T}} (\neg C \sqcup D)$ is in ANF.*

Lemma 4.12. *Let $\mathcal{C}'$ and $\mathcal{T}'$ be a concept and a TBox in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$. Then $\mathcal{C}'$ and $\mathcal{T}'$ can be transformed in polynomial time into $\mathcal{C}$ and $\mathcal{T}$ in ANF such that $\mathcal{C}$ is satisfiable w.r.t. $\mathcal{T}$ if and only if $\mathcal{C}'$ is satisfiable w.r.t. $\mathcal{T}'$.*

Proof. First we describe how negation may be removed from atomic constraints. Let x, y be register names, i, j be natural numbers, and c some integer constant.

- $\neg(S^i x = S^j y)$ can be rewritten as $(S^i x < S^j y) \vee (S^j y < S^i x)$
- $\neg(S^i x = c)$ can be rewritten using a fresh register name z as

$$(S^0 z = c) \wedge ((S^i x < S^0 z) \vee (S^0 z < S^i x))$$

Let $\mathcal{C}'$ and $\mathcal{T}'$ be a concept and a TBox in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$. Let W_{roles}, W_{reg}, and W_{paths} be the sets of role names, register names, and role paths that appear in $\mathcal{C}'$ and $\mathcal{T}'$, respectively. Let d be the maximal depth of path constraints used in $\mathcal{C}'$ and $\mathcal{T}'$.

The proof is split into three parts; in the first part, we propagate the original register values into copy-registers which will make them available locally. In the second part, we use fresh "test" concept names to indicate how the atomic values relate to one another, essentially acting as the logical connectives. Finally, we put it together by rewriting the original concept and TBox into atomic normal form.

Part I In the first step, by relying on the tree model property, we copy in each node u the registers of the ancestors that may occur in the constraints that have the registers of u. For this, we propagate the values in the registers of the ancestors to their children one step at a time. Assume that $W_{\mathsf{reg}} = \{x_1^0, \ldots, x_m^0\}$. For every i where $1 \leq i \leq m$, every k where $1 \leq k \leq d$ and every $P \in W_{\mathsf{paths}}$, we take a fresh register name $x_{i,P}^k$ which will serve as a copy-register. We create a TBox $\mathcal{T}_{\mathsf{prop},d}$ as follows:

$$\mathcal{T}_{\mathsf{prop},d} = \{\top \sqsubseteq \forall r.[\![S^1 x_i^0 = S^0 x_{i,P}^1]\!] \mid r \in W_{\mathsf{roles}}, 1 \leq i \leq m, P \in W_{\mathsf{paths}}\}$$
$$\cup \{\top \sqsubseteq \forall r.[\![S^1 x_{i,P}^k = S^0 x_{i,P}^{k-1}]\!] \mid r \in W_{\mathsf{roles}}, 1 \leq i \leq m, 2 \leq k \leq d, P \in W_{\mathsf{paths}}\}$$

Note that along every path P, the TBox $\mathcal{T}_{\mathsf{prop},d}$ propagates values into copy-registers associated with all the paths in W_{paths}, not just into the copy-registers associated with P. We will later restrict our attention to the appropriate copy-registers depending on context. The next claim follows with a straightforward inductive construction:

Claim 4.13. *Every tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},d}$ contains a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$, and every tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$ can be expanded to a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},d}$.*

Proof. We inductively describe an expansion of a tree model $\mathcal{I}$ of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$ such that the final expansion is a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$. We will simply copy the values in the original registers into their corresponding copy-registers. For copy-registers of elements that are at a smaller depth than the associated path P, we will assign an arbitrary value (namely 0).

1. We first describe an expansion $\mathcal{J}_1$ of $\mathcal{I}$ that will model $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},1}$. For the root element, which is the empty string ε, for every i where $1 \leq i \leq m$, and for every $P \in W_{\mathsf{paths}}$, set
$$(\varepsilon, x^1_{i,P})^{\mathcal{J}_1} = 0.$$
For elements $u, v \in \Delta$ where u is the parent of v, for every i where $1 \leq i \leq m$, and for every $P \in W_{\mathsf{paths}}$, set
$$(v, x^1_{i,P})^{\mathcal{J}_1} = (u, x^0_i)^{\mathcal{I}}.$$
We have that the copy-registers $\{x^1_{i,P} \mid 1 \leq i \leq m, P \in W_{\mathsf{paths}}\}$ are defined for all elements, and the newly assigned register values $\mathcal{J}_1$ satisfy the axioms in $\mathcal{T}_{\mathsf{prop},1}$.

2. We now describe an expansion $\mathcal{J}_{d'}$ given a tree model $\mathcal{J}_{d'-1}$ of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},d'-1}$, where the copy-registers
$$\{x^k_{i,P} \mid 1 \leq i \leq m, P \in W_{\mathsf{paths}}, 1 \leq k \leq d'-1\}$$
are defined for all elements. For the root element ε, for every i where $1 \leq i \leq m$, and for every $P \in W_{\mathsf{paths}}$, set
$$(\varepsilon, x^{d'}_{i,P})^{\mathcal{J}_{d'}} = 0.$$
For elements $u, v \in \Delta$ where u is the parent of v, and for every i where $1 \leq i \leq m$, and for every $P \in W_{\mathsf{paths}}$, set
$$(v, x^{d'}_{i,P})^{\mathcal{J}_{d'}} = (u, x^{d'-1}_{i,P})^{\mathcal{J}_{d'-1}}.$$
We show that $\mathcal{J}_{d'}$ is a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},d'}$. The newly assigned register values satisfy the axioms in $\mathcal{T}_{\mathsf{prop},d'} \setminus \mathcal{T}_{\mathsf{prop},d'-1}$, and $\mathcal{J}_{d'-1}$ satisfies $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},d'-1}$. Since the expansion does not alter previously defined values, and since the register names $x^{d'}_{1,P}, \ldots, x^{d'}_{m,P}$ for $P \in W_{\mathsf{paths}}$ do not appear in $\mathcal{C}'$ nor in $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},d'-1}$, we have that $\mathcal{J}_{d'}$ is a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},d'}$.

Hence $\mathcal{J}_d$ is a tree shaped expansion of $\mathcal{I}$ which satisfies $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop},d}$.

□

We write $\mathcal{T}_{\mathsf{prop}}$ for $\mathcal{T}_{\mathsf{prop},d}$ from now on.

Part II In this step, we create some "test" concept names and axioms that will allow to check whether a given constraint is satisfied in a certain path in a tree model. For $P \in W_{\mathsf{paths}}$ and an atomic constraint Θ, let $\mathsf{loc}(\Theta, P)$ denote the constraint obtained from Θ by replacing each occurrence of $S^j x^0_i$ with $S^0 x^{|P|-j}_{i,P}$. I.e. a reference to an original register at a large depth is replaced with a local reference to its copy-register.

Denote by W_{cnstr} the set of (sub)constraints that appears in $\mathcal{C}'$ or $\mathcal{T}'$. For each $P \in W_{\mathsf{paths}}$ and each $\Theta \in W_{\mathsf{cnstr}}$, take a fresh concept name $T_{P,\Theta}$. For each such P and Θ we add to a TBox $\mathcal{T}_{\mathsf{loc}}$ the following axioms

($\mathbf{A}_1$) $T_{P,\Theta} \equiv T_{P,\Theta_1} \sqcap T_{P,\Theta_2}$ if $\Theta = \Theta_1 \wedge \Theta_2$

($\mathbf{A}_2$) $T_{P,\Theta} \equiv T_{P,\Theta_1} \sqcup T_{P,\Theta_1}$ if $\Theta = \Theta_1 \vee \Theta_2$

($\mathbf{A}_3$) $T_{P,\Theta} \equiv \exists\varepsilon[\![\mathsf{loc}(\Theta, P)]\!]$ if Θ is an atomic constraint.

We first show that the tree models we are interested in can be expanded along with these axioms:

Claim 4.14. *Every tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$ can be expanded to a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$, and every tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$ is a tree model of $\mathcal{C}'$ w.r.t $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$.*

Proof. Let $\mathcal{I}$ be a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$. Let h be the largest circuit-depth of a constraint Θ appearing in $\mathcal{T}'$ or $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}}$. We inductively define $\mathcal{J}^h$ as an expansion of $\mathcal{I}$ by interpreting the fresh concept names of the form $T_{P,\Theta}$.

1. We first describe $\mathcal{J}^0$ by interpreting $T_{P,\Theta}$ for atomic Θ and $P \in W_{\mathsf{paths}}$.

   For $e \in \Delta$, we have $e \in T^{\mathcal{J}^0}_{P,\Theta}$ if and only if $\mathcal{I}, (e) \models \mathsf{loc}(\Theta, P)$. That is, if and only if the copy-registers of e satisfy the localized version of Θ. Recall that loc simply swaps out register names, therefore in $\mathcal{J}^0$ elements may be labeled with $T_{P,\Theta}$ even if they are not the endpoint of a P-path (or even if they are not on a P-path at all), since no information about the path they are on is taken into account in our definition. But we will take care of this later.

   We have that the axioms of the form in item ($\mathbf{A}_3$), which are the only ones relevant in this case, are satisfied by the construction.

2. Let Θ_1, Θ_2 be such that $T_{\Theta_1,P}$ and $T_{\Theta_2,P}$ were interpreted in $\mathcal{J}^{h'-1}$.

   - If $\Theta = \Theta_1 \wedge \Theta_2$ then $e \in T^{\mathcal{J}^{h'}}_{\Theta,P}$ if and only if $e \in T^{\mathcal{J}^{h'-1}}_{\Theta_1,P} \cap T^{\mathcal{J}^{h'-1}}_{\Theta_2,P}$
   - If $\Theta = \Theta_1 \vee \Theta_2$ then $e \in T^{\mathcal{J}^{h'}}_{\Theta,P}$ if and only if $e \in T^{\mathcal{J}^{h'-1}}_{\Theta_1,P} \cup T^{\mathcal{J}^{h'-1}}_{\Theta_2,P}$

   The axioms of $\mathcal{T}_{\mathsf{loc}}$ of the forms in items ($\mathbf{A}_1$) and ($\mathbf{A}_2$), which are the only ones relevant in this case, are satisfied by the semantics of the connectives $\wedge$ and $\vee$.

Therefore we have that $\mathcal{J}^h$ is a tree-shaped expansion of $\mathcal{I}$ that models $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$.

□

Next, we show that $\mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$ indeed relates the satisfaction of constraints along paths to the test concept names.

Claim 4.15. *Let $\mathcal{J}$ be a tree model of $\mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$, and let P be a role path and Θ a constraint appearing in $\mathcal{C}'$ or $\mathcal{T}'$. Then it holds that*

1. *if $e \in T^{\mathcal{J}}_{P,\Theta}$ and $\mathcal{J}$ contains a P-path $e_0, \ldots, e_{|P|}$ that ends at e ($e = e_{|P|}$), then the path $e_0, \ldots, e_{|P|}$ satisfies the constraint Θ in $\mathcal{J}$;*

2. *if $\mathcal{J}$ has a P-path $e_0, \ldots, e_{|P|}$ that satisfies Θ, then $e_{|P|} \in T^{\mathcal{J}}_{P,\Theta}$.*

Notice the qualification in item 1., as a P-path might not exist closer to the root in a tree model.

Proof. First item: let $e_0, \ldots, e_{|P|}$ be a P-path and let $e_{|P|} \in T^{\mathcal{J}}_{P,\Theta}$. Note that from the axioms in $\mathcal{T}_{\mathsf{prop}}$ we have that $(e_j, x^0_{i,P})^{\mathcal{J}} = (e_j, x_i^{|P|-j})^{\mathcal{J}}$ (this is actually true for every $P' \in W_{\mathsf{paths}}$). We proceed by induction on the structure of Θ.

- If Θ is atomic, then by the axioms from item $(\mathbf{A}_3)$ we have $\mathcal{J}, (e_{|P|}) \models \mathsf{loc}(\Theta, P)$. From the fact that $(e_j, x^0_i)^{\mathcal{J}} = (e_j, x^{|P|-j}_{i,P})^{\mathcal{J}}$, together with the definition of loc we get that $\mathcal{J}, (e_0, \ldots, e_{|P|}) \models \Theta$.

- Let Θ_1 and Θ_2 be constraints for which the claim holds. If $\Theta = \Theta_1 \wedge \Theta_2$, then from the axioms in item $(\mathbf{A}_1)$ we have that $e_{|P|} \in (T_{P,\Theta_1} \sqcap T_{P,\Theta_2})^{\mathcal{J}}$. From the IH we have that $\mathcal{J}, (e_0, \ldots, e_{|P|}) \models \Theta_1$ and $\mathcal{J}, (e_0, \ldots, e_{|P|}) \models \Theta_2$ and the claim follows.

- The $\vee$ case follows similarly.

Second item: let a P-path $e_0, \ldots, e_{|P|}$ in $\mathcal{J}$ satisfy Θ. Note that from the axioms in $\mathcal{T}_{\mathsf{prop}}$ we have that $(e_j, x^0_i)^{\mathcal{J}} = (e_j, x^{|P|-j}_{i,P})^{\mathcal{J}}$. We proceed by induction on Θ.

- If Θ is atomic, then from the fact that $(e_j, x^0_i)^{\mathcal{J}} = (e_j, x^{|P|-j}_{i,P})^{\mathcal{J}}$, together with the definition of loc we get that $\mathcal{J}, (e_{|P|}) \models \mathsf{loc}(\Theta, P)$ hence $e_{|P|} \in (\exists \varepsilon. [\![\mathsf{loc}(\Theta, P)]\!])^{\mathcal{J}}$ and from the axioms in item $(\mathbf{A}_3)$ we get that $e_{|P|} \in T^{\mathcal{J}}_{P,\Theta}$.

- Let the claim hold for Θ_1 and Θ_2. If $\Theta = \Theta_1 \wedge \Theta_2$, then by the IH we have that $e_{|P|}$ is in $T^{\mathcal{J}}_{P,\Theta_1}$ and $T^{\mathcal{J}}_{P,\Theta_2}$, hence by semantics of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ we have that $e_{|P|} \in T^{\mathcal{J}}_{P,\Theta_1} \sqcap T^{\mathcal{J}}_{P,\Theta_2}$ and by the axioms in item $(\mathbf{A}_1)$ we have that $e_{|P|} \in T^{\mathcal{J}}_{P,\Theta}$.

- The $\vee$ case follows similarly.

□

Part III In this final step, we use the locally available copy-registers and test concept names to rewrite $\mathcal{C}'$ and $\mathcal{T}'$ into $\mathcal{C}$ and $\mathcal{T}$ in atomic normal form, and use the previously proved claims to show equisatisfiability.

Given a concept D and a (possibly empty) role path $P = r_1 \cdots r_n$, we write $\exists P.D$ meaning

1. the concept $\exists r_1(\exists r_2(\cdots(\exists r_n.D)\cdots))$ when $n > 0$, and
2. the concept D when $n = 0$.

The same notion is defined for $\forall P.D$ in the obvious way.

Let $\mathcal{C}$ and $\mathcal{T}^*$ be obtained from $\mathcal{C}'$ and $\mathcal{T}'$, respectively, by replacing every concept $\exists P.[\![\Theta]\!]$ by $\exists P.T_{P,\Theta}$ and every $\forall P.[\![\Theta]\!]$ by $\forall P.T_{P,\Theta}$. Our desired normalization is the concept $\mathcal{C}$ equipped with the TBox $\mathcal{T} = \mathcal{T}^* \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$.

Let $\mathcal{I}$ be a tree model of $\mathcal{C}'$ and $\mathcal{T}'$. By composing Claim 4.13 and Claim 4.14, we get a tree model $\mathcal{J}$ of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$, to which Claim 4.15 applies. We show that $\mathcal{J}$ is also a tree model of $\mathcal{C}$ w.r.t. $\mathcal{T}$ by showing that $(\exists P.[\![\Theta]\!])^{\mathcal{J}} = (\exists P.T_{P,\Theta})^{\mathcal{J}}$ (showing that $(\forall P.[\![\Theta]\!])^{\mathcal{J}} = (\forall P.T_{P,\Theta})^{\mathcal{J}}$ is similar).

- Let $e_0 \in (\exists P.[\![\Theta]\!])^{\mathcal{J}}$. Then there is a P-path $\vec{e} = (e_0, \ldots, e_{|P|})$ in $\mathcal{J}$ such that $\mathcal{J}, \vec{e} \models \Theta$, therefore by item 2 in Claim 4.15 we have that $e_{|P|} \in T^{\mathcal{J}}_{\Theta,P}$, implying that $e_0 \in (\exists P.T_{\Theta,P})^{\mathcal{J}}$.
- Let $e_0 \in (\exists P.T_{\Theta,P})^{\mathcal{J}}$. Then there exists a P-path $\vec{e} = (e_0, \ldots, e_{|P|})$ in $\mathcal{J}$ such that $e_{|P|} \in T^{\mathcal{J}}_{\Theta,P}$. By item 1 of Claim 4.15, we have that $\mathcal{J}, \vec{e} \models \Theta$ and therefore $e_0 \in (\exists P.[\![\Theta]\!])^{\mathcal{J}}$.

Therefore a tree model $\mathcal{I}$ of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$ can be expanded to a tree model of $\mathcal{C}$ w.r.t. $\mathcal{T}$.

Now we show that every tree model $\hat{\mathcal{J}}$ of $\mathcal{C}$ w.r.t. $\mathcal{T}$ is also a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$. Since $\mathcal{T} \subseteq \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$, Claim 4.15 again applies to $\hat{\mathcal{J}}$. Like before, we have that $(\exists P.[\![\Theta]\!])^{\hat{\mathcal{J}}} = (\exists P.T_{P,\Theta})^{\hat{\mathcal{J}}}$ and $(\forall P.[\![\Theta]\!])^{\hat{\mathcal{J}}} = (\forall P.T_{P,\Theta})^{\hat{\mathcal{J}}}$, therefore $\hat{\mathcal{J}}$ is a tree model of $\mathcal{C}'$ w.r.t. $\mathcal{T}' \cup \mathcal{T}_{\mathsf{prop}} \cup \mathcal{T}_{\mathsf{loc}}$ (and in particular w.r.t. $\mathcal{T}'$). □

4.2.1 Demonstration of the ANF transformation

Here we provide an ANF transformation of a $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concept and TBox as an illustration of the proof above. First, let us name the concepts and constraints:

- C_1 denotes $\exists \varepsilon.[\![\Theta_1]\!]$ where Θ_1 is $S^0 x < S^0 y$
- C_2 denotes $\exists r.[\![\Theta_2]\!]$ where Θ_2 is $S^1 x < S^0 y$

- C_3 denotes $\exists r.[\![\Theta_3]\!]$ where Θ_3 is $\Theta_{31} \wedge \Theta_{32}$, and Θ_{31} is $S^0x < S^1x$ and Θ_{32} is $S^0y = S^1y$
- C_4 denotes $\exists r.[\![\Theta_4]\!]$ where Θ_4 is $\Theta_{41} \wedge \Theta_{42}$, and Θ_{41} is $S^0x < S^1x$ and Θ_{42} is $S^0y < S^1x$

Then consider the concept

$$C_2 \sqcap \exists r.(C_2 \sqcap C_4) \sqcap C_3 \sqcap \exists r.C_3$$

and the TBox $\mathcal{T} = \{\top \sqsubseteq C_1\}$.

Note that the only path appearing in C or $\mathcal{T}$ is r, and that C_1 is already in ANF. We skip the construction of $\mathcal{T}_{\mathsf{prop}}$, and assume that copies of parent register are available in x_r^1 and y_r^1.

Next, by introducing test concept name of the form $T_{r,\Theta}$ we construct $\mathcal{T}_{\mathsf{loc}}$, which contains:

$$\begin{aligned}
T_{r,\Theta_2} &\equiv \exists\varepsilon.[\![S^0x_r^1 < S^0y]\!] \\
T_{r,\Theta_3} &\equiv T_{r,\Theta_{31}} \sqcap T_{r,\Theta_{32}} \\
T_{r,\Theta_{31}} &\equiv \exists\varepsilon.[\![S^0x < S^0x_r^1]\!] \\
T_{r,\Theta_{32}} &\equiv \exists\varepsilon.[\![S^0y = S^0y_r^1]\!] \\
T_{r,\Theta_4} &\equiv T_{r,\Theta_{41}} \sqcap T_{r,\Theta_{42}} \\
T_{r,\Theta_{41}} &\equiv \exists\varepsilon.[\![S^0x < S^0x_r^1]\!] \\
T_{r,\Theta_{42}} &\equiv \exists\varepsilon.[\![S^0y < S^0x_r^1]\!]
\end{aligned}$$

Finally, by replacing the original C_2, C_3, C_4 with their test concept counterparts, we obtain the concept

$$\exists r.T_{r,\Theta_2} \sqcap \exists r.(\exists r.T_{r,\Theta_2} \sqcap (\exists r.T_{r,\Theta_4})) \sqcap \exists r.T_{r,\Theta_3} \sqcap \exists r.(\exists r.T_{r,\Theta_3})$$

and the new TBox $\mathcal{T} \cup \mathcal{T}_{\mathsf{loc}} \cup \mathcal{T}_{\mathsf{prop}}$. Note that since C_1 was already in ANF, the original TBox $\mathcal{T}$ does not change before being added to the final TBox.

Now we are at a point where we can restrict our attention to formulas of this form. Later on, we will need to encode the relationships between the registers as we construct automata for this logic. This would have been technically possible without this normal form, but representing relationships between elements that are a finite-but-unbounded number of hops away from each other would have been extremely cumbersome. But since we can now assume all the path constraints are of length at most 1, the automata construction for this logic will be much more manageable.

CHAPTER 5

Satisfiability Procedure for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$

Here we provide a constructive procedure for deciding satisfiability of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ concepts w.r.t. a general TBox by reducing the satisfiability test to the emptiness of a Rabin tree automaton. We follow the approach of Carapelle and Turhan [38] and split the task into two checks. First we perform an embeddability check for a so-called constraint graph, which is the main difficulty in deciding satisfiability. Loosely speaking, we will need to decide whether a directed tree-like graph, which may also have constant integer labels, can be embedded into $\mathbb{Z}$ in a way that agrees with the linear order induced by its edges and with its constant integer labels. The second check is an ordinary satisfiability check of abstractions of the concept and TBox into plain $\mathcal{ALCF}$, which will be handled in a straightforward way.

The key challenge in the first task is that unsatisfiability can arise from requiring an infinite number of integers to be larger than some integer and smaller than another. For example, if two infinite paths in the graph meet at ever increasing distances. This kind of very non-local behavior is difficult to articulate in a way that is verifiable by automata, as these tend to imply some sort of regularity. So instead of articulating an embeddability condition that is both verifiable by automata and characterizes embeddability of all graphs, we draw great inspiration from Demri and D'Souza [69] and the strategy they used for showing PSpace decidability of constraint LTL. We articulate a Rabin-verifiable embeddability condition, which we prove is necessary in general (Lemma 5.15) and sufficient (Lemma 5.16) for *regular* constraint graphs but not in general. Rabin's Theorem (Theorem 3.14) neatly brings it together as it implies that if *some* graph is embeddable, then also some *regular* graph is embeddable and thus recognized by our automaton. We show that our condition is indeed verifiable by a Rabin tree automaton by spelling out its construction in Section 5.3. This automaton will have a state set of exponential size and a polynomial number of accepting pairs, which will imply an ExpTime upper

bound for concept satisfiability w.r.t. general TBoxes for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ (Theorem 5.40). Our embeddability condition very much resembles the one in [69], however, lifting the approach from the linear structures of LTL to our tree structures involves non-trivial combinatorial graph-theoretic considerations.

5.1 Abstractions and constraint graphs

In order to split the satisfiability check into two tasks, we introduce the notion of abstraction from [38], whose adaptation to our setting results in a very simplified substitution of constraints with concept names. From now on we assume some fixed $\mathcal{C}$ and $\mathcal{T}$ that are in Atomic Normal Form. In particular, this means that the path constraints are of length 1.

Definition 5.1 (Abstraction)**.** *Consider a path constraint $E = \exists r.[\![\Theta]\!]$, where Θ is an atomic constraint. Let $B \in \mathbf{B}$ be a fresh concept name, which we call the* placeholder *of Θ. The* abstraction *of E is defined as*

$$E_a = \exists r.B$$

The abstraction of a universal path constraint is analogous. If $r = \varepsilon$, then the abstraction is simply B.

The abstractions of concepts and TBoxes are the plain $\mathcal{ALCF}$ concepts and TBoxes obtained by replacing all path constraints with their abstracted versions.

Example 5.2. *The abstraction of*

$$\exists r.[\![S^0 x < y]\!] \sqcap \exists s.[\![z = 3]\!]$$

is $\exists r.B_1 \sqcap \exists s.B_2$ where B_1 is a placeholder for $S^0 x < y$ and B_2 is a placeholder for $z = 3$.

Constraint graphs are defined w.r.t. an interpretation of an abstraction, and they indicate the relationships between the register values (smaller/larger than or equal) without using concrete values, as well as equalities of register values with constants. These relationships are expressed via the labels of the vertices where e.g. we may have a placeholder for equality with a constant, and via the edges of the graph where e.g. the origin of a so-called strict edge has a smaller value than its target.

Let $\mathcal{C}_a$, $\mathcal{T}_a$ be the abstractions of $\mathcal{C}$ and $\mathcal{T}$, respectively, and let $\mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ be the set of register names used in $\mathcal{C}$ and $\mathcal{T}$.

Definition 5.3 (Constraint graph)**.** *Let $\mathcal{I}_a = (\Delta^{\mathcal{I}_a}, \cdot^{\mathcal{I}_a})$ be a plain tree-shaped interpretation of $\mathcal{C}_a, \mathcal{T}_a$, that is, an $\mathcal{ALC}$ interpretation as described in Definition 3.5 that is tree-shaped. The constraint graph has the following components. A vertex set $V = \Delta^{\mathcal{I}_a} \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ consisting of pairs of a logical element with each named register,*

a partial labeling $\lambda : V \to 2^{\mathbf{B}}$ *of the vertices with placeholders, and an edge relation* $E = E_< \cup E_=$ *composed of strict edges and equality edges.*

The constraint graph *of* $\mathcal{I}_a$ *is the directed partially labeled graph* $\mathcal{G}_{\mathcal{I}_a} = (V, E, \lambda)$ *where the edge relation is such that, for every* $(v, y), (u, x) \in V$*,*

1. $((v, y), (u, x)) \in E_<$ *if and only if either*
   - $u = v$*, that is, the registers belong to the same logical element,* $v \in B^{\mathcal{I}_a}$ *and* B *is a placeholder for* $S^0 y < S^0 x$*,*
   - u *is the parent of* $v \in B^{\mathcal{I}_a}$ *and* B *is a placeholder for* $S^1 y < S^0 x$*, or*
   - v *is the parent of* $u \in B^{\mathcal{I}_a}$ *and* B *is a placeholder for* $S^0 y < S^1 x$*.*
2. $((v, y), (u, x)) \in E_=$ *if and only if*
   - $u = v$*,* $v \in B^{\mathcal{I}_a}$ *and* B *is a placeholder for* $S^0 y = S^0 x$*,*
   - u *is the parent of* $v \in B^{\mathcal{I}_a}$ *and* B *is a placeholder for* $S^1 y = S^0 x$*, or*
   - v *is the parent of* $u \in B^{\mathcal{I}_a}$ *and* B *is a placeholder for* $S^0 y = S^1 x$*.*

In addition, for a placeholder B *for* $S^0 x = c$*, we have that* $B \in \lambda(u, x)$ *if and only if* $u \in B^{\mathcal{I}_a}$*.*

When the interpretation is clear from context, we write $\mathcal{G}$.

Definition 5.4 (Embeddability into $\mathcal{Z}_c$)**.** *We say a constraint graph* $\mathcal{G}$ *is* embeddable *into* $\mathcal{Z}_c$ *if there is an integer assignment* $\kappa : \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \to \mathbb{Z}$ *to the vertices of* $\mathcal{G}$ *such that for every* $(u, x), (v, y) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$

- *if* $((v, y), (u, x)) \in E_<$ *then* $\kappa(u, x) < \kappa(v, y)$*,*
- *if* $((v, y), (u, x)) \in E_=$ *then* $\kappa(u, x) = \kappa(v, y)$*, and*
- *if* $B \in \lambda(u, x)$ *is a placeholder for* $S^0 x = c$*, then* $\kappa(u, x) = c$*.*

Obviously, any constraint graph that has a cycle in it with a strict edge (a strict cycle) is not embeddable into $\mathbb{Z}$. But it is not always clear from a local view whether a constraint graph is embeddable or not. In Figure 5.1 we see an example of a graph where, loosely speaking, any finite consideration does not indicate whether it is embeddable or not. Note the irregularity of the example; the 'branches' have increasing strict length.

Our reduction of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ satisfiability to two separate checks follows that of Carapelle and Turhan, and is supported by the following theorem:

Theorem 5.5 (Carapelle and Turhan [38])**.** $\mathcal{C}$ *is satisfiable w.r.t.* $\mathcal{T}$ *if and only if there is a tree-shaped* $\mathcal{I}_a \models_{\mathcal{T}_a} \mathcal{C}_a$ *such that* $\mathcal{G}_{\mathcal{I}_a}$ *is embeddable into* $\mathcal{Z}_c$*.*

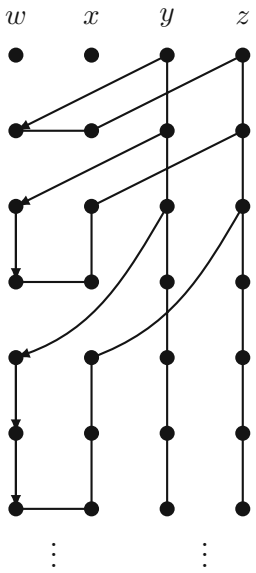


Figure 5.1: An illustration of what a constraint graph might look like. The register names are w, x, y, z, laid out column-wise, and each row holds the registers of a single logical element. Here, non-labeled edges indicate equality between the vertices they connect, and edges with an arrowhead indicate that the origin vertex takes a smaller value than the target vertex. Note that any finite subgraph of this graph is embeddable. However, the whole graph is not; for any difference k between the values assigned to the root element's y and z registers, we will have that the subgraph induced by the elements in the next $k+1$ rows enforces there be more than k different integers between them. The portion of the graph displayed here, for instance, enforces that there are at least 3 integers between the root element's y and z registers.

Since the plain $\mathcal{ALCF}$ satisfiability is not very challenging, the bulk of what follows will focus on the embeddability check, i.e. on how to determine whether a constraint graph that is embeddable into $\mathcal{Z}_c$ exists.

5.1.1 Tree representations of constraint graphs

Our main aim is to devise an embeddability test using tree automata. However, constraint graphs are not exactly trees, so in order to overcome this technical point, we represent constraint graphs using tree decompositions (see Definition 2.4), where each bag holds the subgraph induced by the vertices associated with a logical element and the vertices associated with its parent. We then convert these tree decompositions into tree *representations* using an alphabet where each letter contains the information stored in the tree decomposition bag, without preserving information about the logical element. Note that thanks to our constraints having depth 1, it is enough to consider parent-child pairs. Each letter stores the registers of a parent at the top (top) and the registers of a

child at the bottom (bot), along with the edges and labels from the constraint graph. More precisely, we use two copies x^{top} and x^{bot} of each register $x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$, and call the respective sets $\mathsf{Reg}^{\text{top}}_{\mathcal{C},\mathcal{T}}$ and $\mathsf{Reg}^{\text{bot}}_{\mathcal{C},\mathcal{T}}$.

As we mentioned, the relevant information about equalities with constants is stored as a partial labeling, which we define next. Let c_0 be the smallest integer used in either $\mathcal{C}$ or $\mathcal{T}$ and let c_α be the largest. If no integers were used, set $c_0 = c_\alpha = 0$. Denote by $[c_0, c_\alpha]$ the range of integers between c_0 and c_α, inclusive.

Let

$$\mathbf{U} = \{U_{<c_0}, U_{c_\alpha<}\} \cup \{U_c \mid c \in [c_0, c_\alpha]\}$$

be fresh labels. Let Σ be the set of partially $\mathbf{U}$-labeled graphs where the vertex set is either exactly

$$V = \mathsf{Reg}^{\text{top}}_{\mathcal{C},\mathcal{T}} \cup \mathsf{Reg}^{\text{bot}}_{\mathcal{C},\mathcal{T}} \quad \text{or} \quad V = \mathsf{Reg}^{\text{bot}}_{\mathcal{C},\mathcal{T}}$$

with edge set $E = E_< \cup E_=$. We will sometimes write e.g. $e_=(x, y)$ as shorthand for $(x, y) \in E_=$ and likewise for $E_<$.

Definition 5.6 (Tree representation of constraint graph)**.** *Let $\mathcal{G}$ be the constraint graph of some plain interpretation $\mathcal{I}_a$. For $u \in \Delta^{\mathcal{I}_a}$ with the parent v, define $X(u)$ as the subgraph of $\mathcal{G}$ induced by the registers of u and v, i.e. by:*

$$\{(u, x) \mid x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}\} \cup \{(v, x) \mid x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}\}$$

For the root $u = \varepsilon$, define $X(u)$ as the subgraph of $\mathcal{G}$ induced by $\{(u, x) \mid x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}\}$.

We now remove references to the logical elements. Let $Y(u)$ be the following partially $\mathbf{U}$-labeled graph:

- *The vertices of $Y(u)$ are obtained from $X(u)$ by renaming $(u, x) \mapsto x^{\text{bot}}$ and $(v, x) \mapsto x^{\text{top}}$ for every $x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$. Note that if $u = \varepsilon$, we simply rename $(u, x) \mapsto x^{\text{bot}}$ for every $x \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$.*
- *The edges of $Y(u)$ are exactly those of $X(u)$ (under the renaming).*
- *We have $U_c(x^{\text{bot}})$ if and only if (u, x) is labeled with a placeholder for equality with the constant c, and similarly for x^{top}.*

The tree representation $\mathrm{Tr}(\mathcal{G})$ *of $\mathcal{G}$ is the full Σ-tree where $\mathrm{Tr}(\mathcal{G})(u) = Y(u)$.*

Rather than considering tree representations where nodes are labeled with arbitrary graphs from Σ, it will be convenient to consider trees over a restricted alphabet that contains only graphs that have been enriched with implicit information in a maximal consistent way.

Definition 5.7 (Frame)**.** *A* frame *is a graph in Σ such that:*

1. *there is an edge between every pair of vertices*
2. *there are no strict cycles, i.e. no cycles that include an edge from* $E_<$
3. *equality edges are symmetric, i.e. if* $e_=(x, y)$ *then also* $e_=(y, x)$
4. *every vertex must have exactly one of the labels in* $\mathbf{U}$
5. *if* $e_=(x, y)$ *then* x *and* y *have the same label from* $\mathbf{U}$*, and if* x *and* y *have the same label from* $\mathbf{U} \setminus \{U_{<c_0}, U_{c_\alpha<}\}$ *then* $e_=(x, y)$
6. *if* $e_<(x, y)$ *then either*
   a) $U_{<c_0}(x)$*, or*
   b) $U_{c_\alpha<}(y)$*, or*
   c) $U_{c_i}(x)$ *and* $U_{c_j}(y)$ *with* $c_i, c_j \in [c_0, c_\alpha]$ *and* $c_i < c_j$.

We denote the alphabet of frames by Σ_{fr}.

We will assume that the constraint graphs we are dealing with have been enriched with information such that their tree representations are over Σ_{fr}:

Definition 5.8 (Framified constraint graph)**.** *We say that an augmentation* $\mathcal{G}_{\text{fr}}$ *of a constraint graph* $\mathcal{G}$ *is a* framified constraint graph *if its tree representation is over* Σ_{fr}.

Note that a constraint graph may have multiple framifications; for example, if not all registers are compared to a constant there can be many ways to augment its constraint graph. A constraint graph also may have no framifications; for example, if it contains a strict cycle between the registers of a child and its parent, the corresponding letter in the tree representation will also contain a cycle and thus cannot be augmented so that it is a frame. In fact, there cannot be strict cycles spanning the registers of any number of logical elements in a framification.

Lemma 5.9. *Let* $\mathcal{G}_{\text{fr}}$ *be a framified constraint graph. Then there are no strict cycles in* $\mathcal{G}_{\text{fr}}$.

Proof. We prove the lemma by contradiction. Let p be a strict cycle in $\mathcal{G}_{\text{fr}}$ which spans vertices of exactly k logical elements $u_1, \ldots, u_k$, and assume w.l.o.g. that p starts and ends at u_1, and that u_i is the parent of u_{i+1} for $i \in [k-1]$. If $k \leq 2$, then p is a strict cycle which is contained in the frame $Y(u_2)$, and we reach a contradiction to $\mathcal{G}_{\text{fr}}$ being a framified constraint graph.

Otherwise, consider the restriction p' of p to the vertices of the logical elements u_k and u_{k-1}. Note that we consider two logical elements, as their induced subgraph will be captured as $Y(u_k)$ in the tree representation of $\mathcal{G}_{\text{fr}}$. Let (u_{k-1}, x) be the first vertex on p' and let (u_{k-1}, y) be the last vertex on p'. We show that there exists some edge e from

(u_{k-1}, x) to (u_{k-1}, y). Due to $\mathcal{G}_{\text{fr}}$ being a framified constraint graph, it is enough to show that there is no strict edge from (u_{k-1}, y) to (u_{k-1}, x). Since p' connects (u_{k-1}, x) to (u_{k-1}, y), if there were such a strict edge, there would be a strict cycle in $Y(u_k)$ and we would reach a contradiction to $\mathcal{G}_{\text{fr}}$ being framified. For the same reason, if p' has a strict edge, then e is strict.

By replacing the subpath p' with e in p, we obtain a strict cycle which spans vertices of $k-1$ logical elements. Observe that the strictness is preserved since the potential removal of a strict edge in p' is recovered by e being strict.

Applying this claim inductively, we conclude that there is a strict cycle spanning two logical elements, and reach a contradiction as above.

□

Importantly, an embeddable constraint graph can always be framified by using an assignment κ that witnesses its embeddability, and augmenting the graph with the edges and constant labels implied by it.

Observation 5.10. *Let $\mathcal{G}$ be an embeddable constraint graph. Then there exists a framification of $\mathcal{G}$.*

Also note that in the tree representation of (framified) constraint graphs, the bot part of a vertex coincides with the top parts of its children. We use this fact to restrict our attention from all trees over Σ_{fr} to trees which are, at least conceivably, representations of constraint graphs.

Definition 5.11 (Consistent frame pairs)**.** *Let $\sigma_1, \sigma_2 \in \Sigma_{\text{fr}}$. We call the pair (σ_1, σ_2)* consistent *if the following are equal:*
- *the subgraph induced by the $\mathsf{Reg}^{\text{bot}}_{\mathcal{C},\mathcal{T}}$ vertices of σ_1, and*
- *the result of renaming each x^{top} to x^{bot} in the subgraph induced by the $\mathsf{Reg}^{\text{top}}_{\mathcal{C},\mathcal{T}}$ vertices of σ_2.*

In Figure 5.2 we see three frames and whether the upper frame is consistent with either of the two lower frames.

5.2 Embeddability condition

With the terminology in place, we can turn our attention to articulating an embeddability condition on tree representations of constraint graphs, with the goal of having the condition verifiable with a tree automaton. But first, we must introduce a few more definitions.

In the proofs ahead, we will be talking about paths in trees and constraint graphs, and in particular about paths that only move away from the root. Such paths can be described

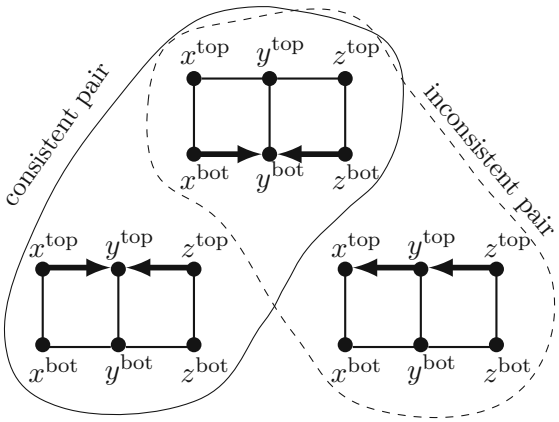


Figure 5.2: The upper frame is consistent with the left hand side lower frame, and not consistent with the right hand side lower frame. The edges which are relevant for whether the pairs are consistent are in bold, and irrelevant edges are only sketched.

by specifying their first node u and which child is visited in each step, i.e. the letter added to the current string representing the path so far.

Definition 5.12 (Path along a word)**.** *Let* $\mathbf{w} = \gamma_1\gamma_2\cdots$ *be a finite or infinite word over* $[n]$ *and let* $u \in [n]^\star$*. In a constraint graph,* a path along $\mathbf{w}$ from (u, x) *is a path* p *where* $p(0) = (u, x)$ *and* $p(i) = (u\gamma_1, x_1)\dots(u\gamma_1\cdots\gamma_i, x_i)$ *for some choice of registers* $x, x_1, \dots, x_i$*.*

The specific registers along the path are immaterial for this definition, and all that is important is the sequence of logical elements visited. The next two definitions are needed for our embeddability condition.

Definition 5.13 (Forward and backward paths)**.** *Let* $\mathcal{G}_{\text{fr}}$ *be a framified constraint graph. An infinite path* $p : \mathbb{N} \to \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ *is a* forward path *if for every* $i \in \mathbb{N}$*, there is an edge from* $p(i)$ *to* $p(i+1)$ *in* $\mathcal{G}_{\text{fr}}$*. It is a* backward path *if for every* $i \in \mathbb{N}$*, there is an edge from* $p(i+1)$ *to* $p(i)$ *in* $\mathcal{G}_{\text{fr}}$*.*

Definition 5.14 (Strict length)**.** *The* strict length *of a finite path* p *in a (possibly framified) constraint graph is the number of strict edges in* p*. For an infinite path* p*, we say that* p is strict *if it has infinitely many strict edges.*

We can now present a condition on framified constraint graphs that will be crucial to deciding embeddability:

(★) There are no $(u, x), (u, y) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ in $\mathcal{G}_{\text{fr}}$ for which we have that: there exists an infinite $\mathbf{w} \in [n]^\omega$, and

1. an infinite forward path f from (u, x) along $\mathbf{w}$, and

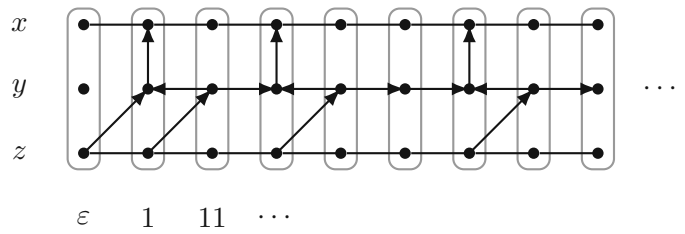


Figure 5.3: A constraint graph satisfying (★) which is not embeddable into $\mathcal{Z}$

2. an infinite backward path b from (u, y) along $\mathbf{w}$

such that f or b is strict, and such that for every $i \in \mathbb{N}$, there is a strict edge from $f(i)$ to $b(i)$.

It is not difficult to show that (★) is a necessary condition for embeddability:

Lemma 5.15. *If a constraint graph is embeddable, then it satisfies the condition* (★).

Proof. Let $(u, x), (v, y) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$. From the definition of embeddability it immediately follows that if there is a finite path from (u, x) to (v, y) of strict length m, then any assignment $\kappa : \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \to \mathbb{Z}$ witnessing the embeddability of $\mathcal{G}_{\text{fr}}$ would satisfy $\kappa((v, y)) - \kappa((u, x)) \geq m$.

Let $\mathcal{G}_{\text{fr}}$ be a framified constraint graph which does not satisfy (★), and let (u, x) and (u, y) be the violating pair. We show that for any natural number m, there is a path from (u, x) to (u, y) of strict length at least m. This is enough, since it would imply there cannot be a register function β which witnesses embeddability: for any determination of the values of $\beta(u, y)$ and $\beta(u, x)$, we will have that there are more than $\beta(u, y) - \beta(u, x)$ integers between them, which of course is not satisfiable. Fix some m and assume w.l.o.g that the forward path f is strict. Then there is a finite prefix p_f of f containing at least m strict edges. Denote the length of p_f by l and let p_b be the l-prefix of the backwards path b. Then the concatenation of p_f with p_b is a path from (u, x) to (u, y), since there is an edge from $f(l)$ to $b(l)$, and it is of strict length at least m.

□

But (★) is not sufficient in general for ensuring embeddability, even for the special case where constraint graphs can be represented by a word over Σ_{fr}. This is demonstrated by the example in Figure 5.3, which is taken from [69]. There is no infinite strict path in the graph, hence (★) is satisfied. However, for any n there is a path from (ε, z) to (ε, x) of strict length at least n, which implies that the constraint graph is not embeddable into $\mathbb{Z}$.

Nonetheless, the condition will allow us to effectively test embeddability, since it is sufficient for *regular* framified constraint graphs, which are those with a regular tree

representation (recall Definition 2.3 of regular trees). The next key lemma is the most technical result in this part of the thesis.

Lemma 5.16. *Let $\mathcal{G}_{\text{fr}}$ be a regular framified constraint graph. If $\mathcal{G}_{\text{fr}}$ satisfies (★), then it is embeddable.*

To prove this lemma, first, we show that if the regular constraint graph $\mathcal{G}_{\text{fr}}$ does not satisfy (★), then there is a pair $(u, x), (u, y)$ such that for every $m \in \mathbb{N}$, there is a path from (u, x) to (u, y) of strict length at least m which only involves vertices whose logical element has the prefix u, that is, the path only involves vertices in the subtree rooted at u. However, the path may move down and up this subtree arbitrarily. We then use framification to extract from this arbitrary path another path p' of a specific shape, which first goes down (away from u) along some w and then goes back up to u. The path p' may have reduced strict length, as this extraction will prune off some subpaths that contain strict edges, but we show a lower bound on the strict length of p' which is a function of m. Then we use regularity to argue that for large enough m, the path p' becomes long enough that it essentially starts repeating itself, thus allowing us to extend it indefinitely (as well as the word it runs along) to obtain the desired forward and backward paths; f is obtained by concatenating the downward portion of p' and b is obtained by concatenating the upward portion. The strict edges between f and b are given by the framification.

We need some definitions and lemmas before starting the proof.

Definition 5.17 (Distance between vertices)**.** *Let $\mathcal{G}$ be a constraint graph, and let $(u, x), (v, z) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ such that there is a path from (u, x) to (v, z) in $\mathcal{G}$. If there is a finite bound on the strict length of paths from (u, x) to (v, z), let m be the maximal strict length of such paths. Then we say the* distance *between (u, x) and (v, z) is m. If there is no finite bound on such paths, we say the distance is* unbounded*.*

Lemma 5.18. *Let $\mathcal{G}_{\text{fr}}$ be a framified constraint graph which is not embeddable into $\mathcal{Z}_c$. Then there exist $(u, x), (v, z) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ such that the distance between (u, x) and (v, z) is unbounded.*

Proof. This is a restatement of a proposition in [70] showing that $\mathcal{Z}_c$ has the EHD-property. The defining formulas (applied to our setting) essentially state that there are no strict cycles (which in our case is given by the framification and Lemma 5.9), and that there exists a bound on the strict length of paths from (u, x) to (v, z), for every (u, x) and (v, z) such that (v, z) is reachable from (u, x). We emphasize that the bound is not global but may vary from pair to pair. □

In the sequel, we freely move from a constraint graph to its tree representation when discussing paths and subtrees for ease of understanding. We also define the following labeling to make later arguments easier to follow:

Definition 5.19 (The labeling ℓ)**.** *Let $\mathcal{G}_{\text{fr}}$ be a framified constraint graph. We define a partial labeling $\ell : \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \to \mathbb{N} \cup \{\infty\}$ on pairs of registers belonging to the same logical element, that is for $(w,x),(w,y) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$, where*

1. *if the largest strict length of a simple path from (w,x) to (w,y) which does not leave the subtree rooted at w is $d \in \mathbb{N}$, then $\ell((w,x),(w,y)) = d$,*
2. *if there is no bound on the strict length of a cycle-free path from (w,x) to (w,y) in the subtree rooted at w, then $\ell((w,x),(w,y)) = \infty$, and*
3. *if there is no path from (w,x) to (w,y) fully contained in the subtree rooted at w, then $\ell((w,x),(w,y))$ is not defined.*

Note that in addition to being only defined on registers of the same logical element, the labeling also only takes into account paths in the subtree rooted at that element. This is in contrast to Definition 5.17, which takes into account all paths. We make some observations about this labeling. In particular, when $\ell((w,x),(w,y)) = 0$, it implies that there *are* paths from (w,x) to (w,y) in the subtree rooted at w, and they are entirely composed of equality edges.

Lemma 5.20. *Let $\mathcal{G}_{\text{fr}}$ be a framified constraint graph which is not embeddable. Then there exist $u \in \Delta$ and $x, y \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ with $\ell((u,x),(u,y)) = \infty$.*

Proof. We first show we can restrict our attention to a single node u, then we show the labeling part of the lemma. $\mathcal{G}_{\text{fr}}$ is not embeddable, therefore by Lemma 5.18 there exist (u',x') and (w',y') with unbounded distance. As the tree representation $\mathcal{G}_{\text{fr}}$ has bounded degree, by König's Lemma we have that there is at least one a subtree in the graph containing infinitely many subpaths of paths from (u',x') to (w',y') of infinitely many strict lengths. Let u be the root of such a subtree such that $|u|$ is minimal in the sense that the previous statement holds for u and does not hold for its parent (if u is not ε). Since we have a bounded number of registers, again by König's Lemma we have that there are registers x, y such that the distance between (u,x) and (u,y) is unbounded. By the minimality of $|u|$ we get that $\ell((u,x),(u,y)) = \infty$.

□

Next, we show that such a pair exists also deep enough in the regular tree such that they are already in the repetitive part:

Definition 5.21. *Let T be a regular tree over Σ. We say $w \in \Sigma^\star$ is* in the repetitive part *of T if there is a strict prefix u of w such that $T|_w = T|_u$.*

Clearly, a vertex who is farther from the root than there are different subtrees will have a subtree which has been seen before on the path from the root.

Observation 5.22. *If T is a regular tree, then any $w \in \Sigma^\star$ of length $|w| > |\{T|_u \mid u \in \Sigma^\star\}|$ is in the repetitive part of T.*

It will be easier, technically, to construct our infinite f and b if our starting point is already in the repetitive part of the tree, so we show it is sound to assume so:

Lemma 5.23. *Let $\mathcal{G}_{\text{fr}}$ be a regular framified constraint graph which is not embeddable into $\mathbb{Z}$. Then there are $(w, x), (w, y) \in \Delta \times \mathsf{Reg}$ in the repetitive part such that $\ell((w, x), (w, y)) = \infty$.*

Proof. We know from Lemma 5.20 that there are some $(u, z_1), (u, z_2) \in \mathsf{Reg}$ such that $\ell((u, z_1), (u, z_2)) = \infty$. By the definition of ℓ and the fact that we have finite degree, by König's Lemma we have that u has a child u' and there exist registers z_1', z_2' such that $\ell((u', z_1'), (u', z_2')) = \infty$. We apply this argument inductively until we reach the repetitive part, which by Observation 5.22 is a finite number of times. □

For our proof it will be enough to consider partial framifications of constraint graphs, since we will not rely on every existing edge. Observe that due to the inability of framifications to introduce strict cycles (Lemma 5.9), all the framifications of a constraint graph $\mathcal{G}$ contain a common subgraph whose edges relate to ℓ in the following way:

Observation 5.24. *Let $\mathcal{G}_{\text{fr}}$ be a framification of $\mathcal{G}$. Then for every $u \in \Delta$ and $x, y \in \mathsf{Reg}$, we have in $\mathcal{G}_{\text{fr}}$:*

1. *An equality edge $e_=((u, x), (u, y))$ if $\ell((u, x), (u, y)) = 0$*
2. *A strict edge $e_<((u, x), (u, y))$ if $\ell((u, x), (u, y)) \in \mathbb{N}^+ \cup \{\infty\}$*

Note that the *maximal* common subgraph may contain additional edges, as ℓ only takes into account paths in the subtree rooted at a vertex, but for our proofs, we will only use these common edges and they will suffice.

We introduce one more definition before we begin the proof, in order to make speaking about our paths easier:

Definition 5.25 (Downward and upward trend)**.** *We say a path p in a constraint graph has a* downward trend *if the elements $w \in \Delta$ along p have (strictly) increasing depth. Similarly, a path has an* upward trend *if the elements have decreasing depth.*

Essentially, as long as a path is fully contained in a subtree, it goes down if it moves farther from the root of the full tree and it goes up if it moves closer to the root. We will be interested in constructing a path which changes its trend once, which is dubbed an down-then-up path:

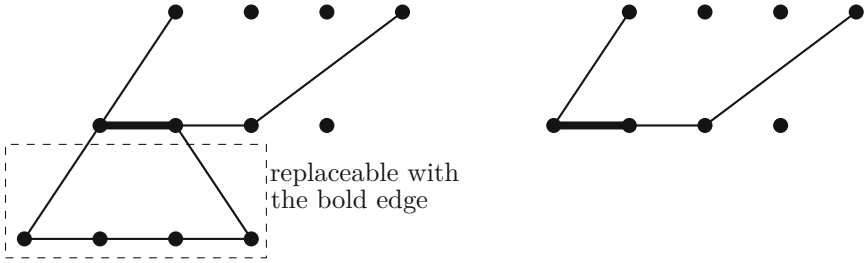


Figure 5.4: All edges in the graph are equality edges. We can replace the equality path with a single edge (in bold) without reducing the strict length of encompassing paths

Definition 5.26. *Let $(w,x),(w,y) \in \Delta \times \mathsf{Reg}$. We say a path from (w,x) to (w,y) in a constraint graph goes* down-then-up *if it can be broken into two contiguous subpaths where the first subpath has a downward trend and the second one has an upward trend.*

Lemma 5.27. *Let $\mathcal{G}_{\text{fr}}$ be a framified constraint graph. If $(u,x),(u,y) \in \Delta \times \mathsf{Reg}$ are such that $\ell((u,x),(u,y)) = \infty$, then for every $n \in \mathbb{N}$ there is a down-then-up path p' from (u,x) to (u,y) in $\mathcal{G}_{\text{fr}}$ of strict length at least n.*

Proof of Lemma 5.27

There are two parts to the proof. First we describe, given a path p from (u,x) to (u,y), another path p' from (u,x) to (u,y) which goes down-then-up. In the second part, we give a lower bound on the strict length of the new path p' as a function of the strict length of the original path p. Later, we will argue that a path with large enough strict length will allow us to construct the desired f and b of ($\bigstar$).

Let $u \in \Delta$ and $x, y \in \mathsf{Reg}$ such that $\ell((u,x),(u,y)) = \infty$ and let p be a path in $\mathcal{G}_{\text{fr}}$ from (u,x) to (u,y) of strict length $\geq N$ and assume p has no cycles. Denote by d the maximal depth of p.

Constructing p' We begin with some simple observations:

Observation 5.28. *We may assume that any subpath p'' of p which begins and ends with vertices of the same logical element has strict length at least 1, otherwise due to framification we have an equality edge e'' between the start and end of p''. Then we may consider the path where p'' is replaced by e'', which has the same strict length as the original path p.*

See Figure 5.4 for an example where this observation applies.

Observation 5.29. *The number of times we may see a certain $w \in \Delta$ along p is bounded by the number of registers $|\mathsf{Reg}_{\mathcal{C},\mathcal{T}}|$, since we assume no cycles.*

Assume that the logical element u of the starting and ending points of p appears exactly twice along p; at its beginning and its end. We inductively construct $\text{pre}^{(i)}$, $\text{mid}^{(i)}$, and $\text{suf}^{(i)}$, where $\text{pre}^{(i)}$ has a downward trend, $\text{suf}^{(i)}$ has an upward trend, and $\text{mid}^{(i)}$ is a subpath of the original p, which will be manipulated in later steps. For this construction, we will use the edges promised by the framification, and the ultimate construction will be a down-then-up path.

For a path q with endpoints a, b, we denote by $q \setminus \{a, b\}$ the subpath of q obtained by excluding a and b.

Set

$$\text{pre}^{(0)} = (u, x), \quad \text{suf}^{(0)} = (u, y), \quad \text{mid}^{(0)} = p \setminus \{(u, x), (u, y)\}$$

Note that $\text{pre}^{(0)}$ and $\text{suf}^{(0)}$ indeed have a strict downward and upward trend, respectively, and that $\text{mid}^{(0)}$ begins and ends with vertices associated with the same logical element.

Given $\text{pre}^{(i)}$, $\text{suf}^{(i)}$, and $\text{mid}^{(i)}$, we define $\text{pre}^{(i+1)}$, $\text{suf}^{(i+1)}$, and $\text{mid}^{(i+1)}$. Denote the logical element appearing in the first and last vertices on $\text{mid}^{(i)}$ by u_i. How we proceed depends on how many times u_i is visited in $\text{mid}^{(i)}$. If it is twice or less, how we proceed is simple since there are no multiple trend changes touching u_i. If it is more than twice, we will use the edges promised by the framification to take shortcuts that eliminate the trend changing parts of $\text{mid}^{(i)}$ that touch u_i. More precisely:

1. If u_i appears exactly twice on $\text{mid}^{(i)}$, denote its appearances by (u_i, x_i) and (u_i, y_i). Then define
$$\begin{aligned} \text{pre}^{(i+1)} &= \text{pre}^{(i)}(u_i, x_i), \\ \text{suf}^{(i+1)} &= (u_i, y_i)\text{suf}^{(i)}, \\ \text{mid}^{(i+1)} &= \text{mid}^{(i)} \setminus \{(u_i, x_i), (u_i, y_i)\}. \end{aligned}$$

2. If u_i appears more than twice on $\text{mid}^{(i)}$, let (u_i, x_i) and (u_i, y_i) be the pair of subsequent appearances of u_i on $\text{mid}^{(i)}$ that enclose a subpath with the largest strict length (if there are multiple such pairs, take the earliest one).

   a) Define
   $$\begin{aligned} \text{pre}^{(i+1)} &= \text{pre}^{(i)}(u_i, x_i), \\ \text{suf}^{(i+1)} &= (u_i, y_i)\text{suf}^{(i)}. \end{aligned}$$

   Since there is a path from $\text{pre}^{(i)}$ to (u_i, x_i), due to framification, there is an edge e from the end of $\text{pre}^{(i)}$ to (u_i, x_i), and similarly there is an edge e' from (u_i, y_i) to the beginning of $\text{suf}^{(i)}$. Therefore, $\text{pre}^{(i+1)}$ and $\text{suf}^{(i+1)}$ are well defined. Furthermore, since at least of one these edges acts as a shortcut replacing a strict path (we may assume it is a strict path due to Observation 5.28, as it begins and ends in the same logical element), at least one of these edges is strict.

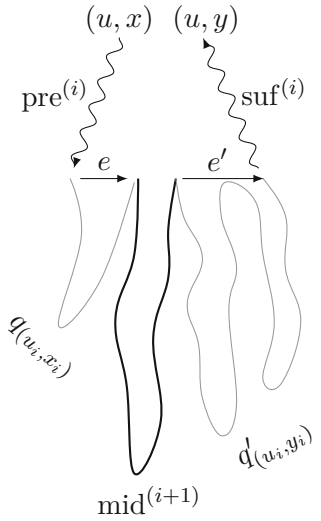


Figure 5.5: What the temporary path may look like during the inductive construction. Here, $\text{mid}^{(i)}$ is the concatenation of the paths $q_{(u_i,x_i)}$, the subpath drawn in bold, and $q'_{(u_i,y_i)}$. In order to progress toward a down-then-up path, $q_{(u_i,x_i)}$ will be replaced by the strict edge e, $q'_{(u_i,y_i)}$ will be replaced by the strict edge e', and $\text{mid}^{(i+1)}$ will be the subpath drawn in bold. The new prefix $\text{pre}^{(i+1)}$ will be the concatenation of $\text{pre}^{(i)}$ with e, and $\text{suf}^{(i+1)}$ will be the concatenation of $\text{suf}^{(i)}$ to e'.

b) We use the edges e and e' described above to shortcut the multiple paths visiting the logical element u_i, and connect $\text{pre}^{(i+1)}$ to $\text{mid}^{(i+1)}$, and $\text{mid}^{(i+1)}$ to $\text{suf}^{(i)}$ as follows. Let $q_{(u_i,x_i)}$ be the subpath of $\text{mid}^{(i)}$ beginning at the first vertex of $\text{mid}^{(i)}$ and ending at (u_i, x_i). Let $q'_{(u_i,y_i)}$ be the subpath of $\text{mid}^{(i)}$ beginning at (u_i, y_i) and ending at the last vertex of $\text{mid}^{(i)}$. Then define

$$\text{mid}^{(i+1)} = \text{mid}^{(i)} \setminus \{q_{(u_i,x_i)}, q'_{(u_i,y_i)}\}.$$

I.e. $\text{mid}^{(i+1)}$ is the result of removing the subpaths that were replaced by the shortcuts e and e'. See Figure 5.5 for an illustration.

3. If $\text{mid}^{(i)}$ is empty, then define

$$\begin{aligned} \text{pre}^{(i+1)} &= \text{pre}^{(i)}, \\ \text{suf}^{(i+1)} &= \text{suf}^{(i)}, \\ \text{mid}^{(i+1)} &= \text{mid}^{(i)}. \end{aligned}$$

The following observation will support there being a strict edge between every node on f to every node on b:

Observation 5.30. *If* $\text{mid}^{(i)}$ *has strict length* > 1*, then due to framification, there is a strict edge from* (u_i, x_i) *to* (u_i, y_i)*.*

Recall that the starting point is at depth d. Therefore after at most d steps of the above construction, we will have $\text{mid}^{(d)} = \varepsilon$. Take $p' = \text{pre}^{(d)}\text{suf}^{(d)}$. We have that for every depth, an element of that depth appears at most twice in p' (in fact, exactly twice except for possibly the deepest element).

A lower bound on the strict length of p' As we used shortcut edges to transform the constructed path to a down-then-up path, some strict edges may have been lost, and so we need to argue about the strict length of p; Note that only applications of case 2 decrease the strict length of p'. Therefore the strict length of p' will be the smallest when its construction involves the most applications of case 2. We want to bound the number of times case 2 can be applied before we reach $\text{mid}^{(i)} = \varepsilon$.

Recall that n is the degree of the tree representation of $\mathcal{G}_{\text{fr}}$. For $0 \leq i \leq d$, denote by $N^{(i)}_{\text{mid}}$ the strict length of $\text{mid}^{(i)}$. Denote by ρ the number of register names used, i.e. $|\mathsf{Reg}_{\mathcal{C},\mathcal{T}}|$.

Assume that we apply case 2 in step i, meaning u_i appears more than twice on $\text{mid}^{(i)}$. Due to the degree being n, this implies that there is a pair (u_i, x_i) and (u_i, y_i) of subsequent appearances whose subpath has strict length at least $(N^{(i)}_{\text{mid}} - \rho)/n$. We subtract ρ in order to account for possibly losing ρ strict edges within the same depth as we perform Step 2b. In other words,

$$N^{(i+1)}_{\text{mid}} \geq (N^{(i)}_{\text{mid}} - \rho)/n$$

Let us define this bound of $N^{(i)}_{\text{mid}}$ from below as a series.

$$\begin{aligned} a_0 &= N \\ a_1 &= \frac{1}{n}(a_0 - \rho) \\ a_{i+1} &= \frac{1}{n}(a_i - \rho) \end{aligned}$$

Claim 5.31. *For $m \geq 1$,*

$$a_m = \frac{N - \rho}{n^m} - \rho \sum_{h=1}^{m-1} n^{-h}$$

Proof. By induction on m.

- We show the claim holds for $m = 1$: By definition, we have

  $$a_1 = \frac{1}{n}(a_0 - \rho) = \frac{N - \rho}{n}$$

  By substituting 1 for m, we have:

  $$\frac{N - \rho}{n^m} - \rho \sum_{h=1}^{m-1} n^{-h}\Bigg|_{m=1} = \frac{N - \rho}{n}$$

- We assume the claim holds for $m = m'$:

$$a_{m'} = \frac{N-\rho}{n^{m'}} - \rho \sum_{h=1}^{m'-1} n^{-h}$$

- We show correctness for $m = m' + 1$:

$$\begin{aligned} a_{m'+1} &= \frac{a_{m'} - \rho}{n} \\ &= \frac{N-\rho}{n^{m'+1}} - \rho \sum_{h=1}^{m'-1} n^{-h-1} - \frac{\rho}{n} \\ &= \frac{N-\rho}{n^{m'+1}} - \rho \sum_{h=2}^{(m'+1)-1} n^{-h} - \frac{\rho}{n} \\ &= \frac{N-\rho}{n^{m'+1}} - \rho \sum_{h=1}^{(m'+1)-1} n^{-h} \end{aligned}$$

□

Since this series bounds $N_{\text{mid}}^{(m)}$ from below, we have that

$$N_{\text{mid}}^{(m)} \geq \frac{N-\rho}{n^m} - \rho \sum_{h=1}^{m-1} n^{-h}$$

Furthermore, we have that

$$\frac{N-\rho}{n^m} - \rho \sum_{h=1}^{m-1} n^{-h} > \frac{N-\rho}{n^m} - \rho \sum_{h=1}^{\infty} n^{-h} = \frac{N-\rho}{n^m} - \rho \frac{1}{n-1}$$

We solve for m in order to bound the maximal number of times case 2 may be applied in the construction of p'.

$$\frac{N-\rho}{n^m} - \rho \frac{1}{n-1} = 0$$

After some algebra we get

$$m = \log_n\left(\frac{(N-\rho)(n-1)}{\rho}\right)$$

To recap – the constructed path p' has the smallest strict length if case 2 was applied a maximal number of times, and we have showed that this may occur at most $\log_n(\frac{(N-\rho)(n-1)}{\rho})$ times. However, since each time we apply case 2 we have at least one strict edge added to the path (in Step 2a), we also have at least $\log_n(\frac{(N-\rho)(n-1)}{\rho})$ strict edges in p'. Obviously,

$$\lim_{N\to\infty} \left(\log_n\left(\frac{(N-\rho)(n-1)}{\rho}\right)\right) = \infty,$$

and we have our proof of Lemma 5.27.

Back to the proof of Lemma 5.16. Finally, we can prove the lemma which will facilitate the construction of the paths violating (★). Let N_{st} be the number of different subtrees in the tree representation of $\mathcal{G}_{\text{fr}}$. Let $N_{\text{rep}} = \rho^4 N_{\text{st}} + 1$.

Lemma 5.32. *Let $u \in \Delta$ be in the repetitive part of $\mathcal{G}_{\text{fr}}$ and let $x, y \in \mathsf{Reg}$ such that $\ell((u,x),(u,y)) = \infty$. Let p be the path promised by Lemma 5.27 of strict length $2N_{\text{rep}}$. Then there is $v \in \Delta$ on p and registers z, z' such that $(v,z),(v,z')$ violate (★).*

Proof. Since p is a down-then-up path of strict length $2N_{\text{rep}}$, there are at least N_{rep} strict edges in one of the directions on p. Assume w.l.o.g. that it is the downward direction.

Since the number of strict edges in the downward direction is larger than the number of possible combinations of a subtree with a quadruple of registers, we have the following on p (see Figure 5.6):

1. logical elements v and $v' = vw$ for some $w \in [n]^+$ such that v and v' have isomorphic subtrees,
2. registers z, z' such that there is a path f_1 from (v,z) to (v',z) with strict length at least 1, and a path b_1 from (v',z') to (v,z').

Since p is a path from (u,x) to (u,y) which goes through (v',z) and then (v',z'), and since framifications may not introduce strict cycles (Lemma 5.9), we have an edge e' from (v',z) to (v',z'). Since v and v' have isomorphic subtrees, this implies there is also an (isomorphic) edge e from (v,z) to (v,z'). Finally, since f_1 is strict, by Observation 5.24 we have that the edges e and e' are strict.

As v and v' have isomorphic subtrees, this implies that there is an infinite strict forward path f from (v,z), since the strict forward path f_1 can be concatenated indefinitely. Similarly, there is an infinite backward path b into (v,z'), as the path b_1 can also be concatenated.

It remains to show that there is strict edge from $f(i)$ to $b(i)$ for every $i \geq 0$. By our construction, we have that for every i, there is a strict path from $f(i)$ to $b(i)$ – for example one which uses a copy of the edge e above. Furthermore, by the construction in the proof of Lemma 5.27, for every i, $f(i)$ and $b(i)$ are vertices associated with the same logical element. Therefore by Observation 5.30 we have a strict edge from $f(i)$ to $b(i)$.

□

5.3 Automata for deciding satisfiability

Since condition (★) is necessary and sufficient for the embeddability of regular framified constraint graphs, from Rabin's Theorem (Theorem 3.14) we get:

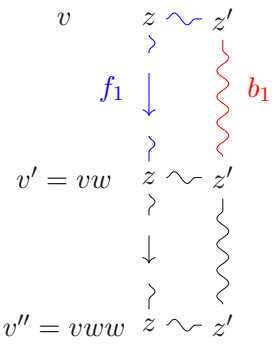


Figure 5.6: Subgraph of the constraint graph. v, v', v'' have isomorphic subtrees along a periodic word w, f_1 is a forward path with at least one strict edge and b_1 is a backward path. Note that f_1 goes from the z register of a vertex to the z register of an isomorphic vertex (and b_1 behaves similarly).

Lemma 5.33. *Let $\mathcal{A}_{\text{emb}}$ be a Rabin tree automaton that accepts exactly the consistent trees over Σ_{fr} satisfying (★). There is an embeddable constraint graph if and only if $\mathscr{L}(\mathcal{A}_{\text{emb}}) \neq \emptyset$.*

Proof. If there is an embeddable constraint graph $\mathcal{G}$, then it has some framification $\mathcal{G}_{\text{fr}}$ (Observation 5.10), which satisfies the condition (★) (Lemma 5.15). Therefore the tree representation of $\mathcal{G}_{\text{fr}}$ is accepted by $\mathcal{A}_{\text{emb}}$ and $\mathscr{L}(\mathcal{A}_{\text{emb}}) \neq \emptyset$. For the other direction, assume $\mathscr{L}(\mathcal{A}_{\text{emb}}) \neq \emptyset$. Then by Rabin's Theorem, there is a regular tree $T \in \mathscr{L}(\mathcal{A}_{\text{emb}})$, which satisfies the condition (★). By Lemma 5.16, we have that the constraint graph represented by T is embeddable. □

Therefore it remains to show that the condition (★) is indeed verifiable by a Rabin tree automaton (See Section 3.3 for a reminder). We do this next.

5.3.1 Checking consistency of trees.

In our constructions of automata, it is useful to assume that they run on trees over Σ_{fr} that are consistent (in the sense of Definition 5.11), rather than complicating the constructions by incorporating the consistency check. Therefore we first describe an automaton $\mathcal{A}_{\text{ct}}$ which accepts exactly the consistent trees, which we later intersect with the appropriate automata (using the product automaton construction in the proof of Lemma 3.13). The automaton $\mathcal{A}_{\text{ct}}$ simply verifies the conditions of Definition 5.11 by only having transitions between consistent pairs of frames, and making sure the root vertex is labeled with a frame whose vertex set consists exactly of $\mathsf{Reg}^{\text{bot}}_{\mathcal{C},\mathcal{T}}$.

First we denote the pairs of consistent pairs of frames as in Definition 5.11:

$$\text{CFr} = \{(\sigma_1, \sigma_2) \in \Sigma_{\text{fr}} \times \Sigma_{\text{fr}} \mid (\sigma_1, \sigma_2) \text{ is consistent}\}$$

Also denote the set of frames whose vertex set only has bot vertices:

$$\Sigma_{\text{fr}}^{\text{bot}} = \{\sigma \in \Sigma_{\text{fr}} \mid V(\sigma) = \mathsf{Reg}_{\mathcal{C},\mathcal{T}}^{\text{bot}}\}.$$

We define $\mathcal{A}_{\text{ct}} = (Q, q_0, \delta_{\text{ct}}, (\emptyset, Q))$, where:

- $Q = \{q_0\} \cup \{q_\sigma \mid \sigma \in \Sigma_{\text{fr}}\}$.
- For every $\sigma \in \Sigma_{\text{fr}}^{\text{bot}}$, we have $(q_0, \sigma) \ni (q_\sigma, \ldots, q_\sigma)$.
- For every $(\sigma_1, \sigma_2) \in \text{CFr}$, we have $(q_{\sigma_1}, \sigma_2) \ni (q_{\sigma_2}, \ldots, q_{\sigma_2})$.

Proposition 5.34. *The Rabin tree automaton $\mathcal{A}_{\text{ct}}$ accepts exactly the consistent trees over Σ_{fr}.*

Observation 5.35. *The number of states of $\mathcal{A}_{\text{ct}}$ is exponential in the size of $\mathcal{C}$ and $\mathcal{T}$, and its Ω has one pair.*

5.3.2 A Rabin tree automaton for verifying (★)

It will be easier to first construct an automaton which finds a violating pair and then complement it. Therefore we describe an automaton $\mathcal{B}$ which runs on consistent trees over Σ_{fr}, and finds a pair of registers which violates (★). The desired $\mathcal{A}_{\text{emb}}$ is the intersection of the complement of $\mathcal{B}$ with $\mathcal{A}_{\text{ct}}$, the consistency checking automaton.

We define $\mathcal{B}$ while simultaneously describing its behavior.

We let $\mathcal{B} = (Q, q_0, \delta, (\emptyset, U))$ with

- $Q = \{q_0, q_1, q_2\} \cup Q_p$ where Q_p is the set of *path states*

  $$Q_p = \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \times \{f, b\} \times \{0, 1\}.$$

  The flags in the path states will indicate whether it is the forward path f or the backward path b which is strict, and the binary flag will indicate whether the automaton just encountered a strict edge in the direction that was guessed to be strict. We need to make note of these strict edges in order to verify that the path is indeed strict (has infinitely many strict edges).

- We describe δ next. First, $\mathcal{B}$ travels down the tree until it reaches a node that it guesses has the violating pair. Therefore we have that the initial state q_0 and the state q_2 both represent that the violating pair is (a) in the current subtree, (b) but not in the current node. The reason for there being two states which serve the same function is the technicality with Rabin automata having a single initial state which is only visited at the root of the input tree.

  Therefore $\mathcal{B}$ needs to pick in which subtree to continue its search (which would also imply that it discards the other choices). Thus from either of these states, $\mathcal{B}$

picks one child for which (a) is also true, and possibly also (b). In the latter case, it moves to q_2 for that child, while the other children go into q_1. State q_1 means that the problematic pair is not in the subtree, and once $\mathcal{B}$ visits some node in q_1, it stays in q_1 for all its descendants.

Precisely speaking, denote by $\mathbf{q}_i^e$ the n-tuple containing q_2 for entry i and q_1 for every other entry.

- For every $i \in [n]$ and $\sigma \in \Sigma_{\text{fr}}$ we have

$$(q_0, \sigma) \ni \mathbf{q}_i^e \qquad \text{and} \qquad (q_2, \sigma) \ni \mathbf{q}_i^e$$

- For every $\sigma \in \Sigma_{\text{fr}}$, we have $(q_1, \sigma) \ni (q_1, \ldots, q_1)$

At some point, $\mathcal{B}$ moves from a node where both (a) and (b) are true (that is, q_0 or q_2) to a node where (b) no longer holds, i.e., it guesses that the violating pair is in that node u. Then, it guesses the registers x, y as the problematic pair and whether it is the forward path f or the backward path b which will be strict. This will be stored in the flag f or b, which once chosen cannot change during the run.

If the guessed pair x, y has a $<$ relation (needed for the strict edge from $f(0)$ to $b(0)$ required by ($\bigstar$)), $\mathcal{B}$ transitions accordingly to a path state $(x, y, f, 0)$ or $(x, y, b, 0)$. To make this precise, we introduce more notation. For every $i \in [n]$, $h \in \{f, b\}$, and $- \in \{0, 1\}$, denote by $(x, y, h, -)_i$ the n-tuple containing $(x, y, h, -)$ for entry i and q_1 for every other entry.

- For every $i \in [n]$ and $h \in \{f, b\}$, if $e_<(x^{\text{bot}}, y^{\text{bot}}) \in \sigma$ we have

$$(q_0, \sigma) \ni (x, y, h, 0)_i \qquad \text{and} \qquad (q_2, \sigma) \ni (x, y, h, 0)_i$$

Now, in every step, $\mathcal{B}$ attempts to expand f and b by guessing a child v and a new pair z, w with a strict edge between z and w. It moves to a path state for the child v indicating z and w, and when doing so, it also uses the other binary flag to indicate whether $\mathcal{B}$ just witnessed a strict edge relevant to f or b (flag value 1), or not (flag value 0). For the other children, it moves to state q_1.

We describe the transitions for the case where the forward path is strict; there are similar transitions for backward paths. If the guess correctly extends f and b, that is,

$$e_<(z^{\text{bot}}, w^{\text{bot}}) \in \sigma \qquad \text{and} \qquad e_<(y^{\text{top}}, w^{\text{bot}}) \notin \sigma$$

then, for every $i \in [n]$,

- if the current edge on the forward path is strict, that is, $e_<(x^{\text{top}}, z^{\text{bot}}) \in \sigma$, we have

$$((x, y, f, -), \sigma) \ni (z, w, f, 1)_i$$

- and if the current edge is not strict, that is, $e_=(x^{\text{top}}, z^{\text{bot}}) \in \sigma$, then we have

$$((x, y, f, -), \sigma) \ni (z, w, f, 0)_i$$

- Paths looping in q_1 are successful, and to guarantee that the guessed path is strict, it must contain infinitely many strict edges (marked with flag 1). Therefore we set

$$U = \{q_1\} \cup \{(x, y, h, 1) \mid h \in \{f, b\}\}$$

As mentioned before, the automaton $\mathcal{A}_{\text{emb}}$ is the complement automaton of $\mathcal{B}$ intersected with $\mathcal{A}_{\text{ct}}$. It has the same alphabet, but it may have exponentially many more states (Theorem 3.12).

Proposition 5.36. *The Rabin tree automaton $\mathcal{A}_{\text{emb}}$ accepts exactly the consistent trees over Σ_{fr} that satisfy* (★).

Observation 5.37. *The number of states of $\mathcal{B}$ is polynomial in the size of $\mathcal{C}$ and $\mathcal{T}$, its Ω has one pair, and the alphabet is exponential. Therefore $\mathcal{A}_{\text{emb}}$ has a number of states and alphabet exponential in the size of $\mathcal{C}$ and $\mathcal{T}$, and its Ω has a constant number of pairs.*

5.3.3 Satisfiability of the abstracted $\mathcal{ALCF}$ part

We use the automata construction of Section 3.4 and reduce satisfiability of the abstract $\mathcal{C}_a$ w.r.t. $\mathcal{T}_a$ to the emptiness of the Rabin tree automaton $\mathcal{A}_{\text{alcf}}$ that runs on trees over the alphabet Ξ of Hintikka sets (Definition 3.15). Note that, for completeness, it is important that the automaton accepts *all* tree models, as opposed to e.g. accepting some canonical model which may not necessarily have an embeddable constraint graph. Furthermore, we note that since $\mathcal{C}_a$ and $\mathcal{T}_a$ have size polynomial in $\mathcal{C}$ and $\mathcal{T}$, the following still holds:

Observation 5.38. *The alphabet Ξ and the number of states of $\mathcal{A}_{\text{alcf}}$ are exponential in the size of $\mathcal{C}$ and $\mathcal{T}$, and the Ω of $\mathcal{A}_{\text{alcf}}$ has one pair.*

5.3.4 Matching the alphabets

The final automaton $\mathcal{A}_{\mathcal{T},\mathcal{C}}$ should accept only representations of models of the abstraction of $\mathcal{C}$ and $\mathcal{T}$ whose constraint graph is embeddable. The embeddability check is done by $\mathcal{A}_{\mathsf{Emb}}$ over the alphabet Σ_{fr}, and the satisfiability check of the $\mathcal{ALCF}$ part is done by $\mathcal{A}_{\text{alcf}}$ over the alphabet Ξ, therefore we modify both automata to use the same alphabet. We let $\mathcal{A}'_{\mathsf{Emb}}$ and $\mathcal{A}'_{\text{alcf}}$ be the modification of $\mathcal{A}_{\mathsf{Emb}}$ and $\mathcal{A}_{\text{alcf}}$ to trees over the product alphabet $\Sigma_{\text{fr}} \times \Xi$, while completely ignoring the irrelevant part of each letter. Obviously, this is merely a syntactic change, and the state sets of $\mathcal{A}'_{\mathsf{Emb}}$ and $\mathcal{A}'_{\text{alcf}}$ are not affected and remain exponential in $|\mathcal{C}, \mathcal{T}|$, nor are their Ω sets, which still have a constant number of pairs.

We still need one more ingredient. It is not enough to verify if a tree over $\Sigma_{\text{fr}} \times \Xi$ is accepted by $\mathcal{A}'_{\text{emb}}$, which ignores Ξ, and by $\mathcal{A}'_{\text{alcf}}$, which ignores Σ_{fr}: such a tree could pair a model of the abstraction with a totally unrelated constraint graph. In order to verify that the constraint graph is the one induced by the interpretation of the abstraction, we take an automaton $\mathcal{A}_{\text{m}}$ that considers both parts of the product alphabet $\Sigma_{\text{fr}} \times \Xi$, and

accepts the trees where the restriction of the input to Ξ, corresponding to the abstraction to $\mathcal{ALCF}$, induces the constraint graph corresponding to the restriction of the input to Σ_{fr}.

Below, this is done by verifying the conditions described in Definition 5.3 while applying the placeholders in the Ξ part of the letter to the bot vertices in the Σ_{fr} part of the letter. Such a test is built into the transition relation, using a constant number of states. In order to verify that the graph induced by the Ξ part of the letter is contained in the framification of the $\mathsf{Reg}^{\text{bot}}_{\mathcal{C},\mathcal{T}}$ vertices in the Σ_{fr} part, we introduce some notation. Recall that $\mathbf{B}$ is the set of placeholders introduced during the abstraction of $\mathcal{C}$ and $\mathcal{T}$. For $\xi \in \Xi$, denote $\mathbf{B}(\xi) = \xi \cap \mathbf{B}$, i.e. the set of placeholders appearing in ξ. For $\sigma \in \Sigma_{\text{fr}}$ denote

$$\mathbf{B}_{\text{bot}}(\sigma) = \{B \in \mathbf{B} \mid \text{ there is } v \in \mathsf{Reg}^{\text{bot}}_{\mathcal{C},\mathcal{T}} \text{ in } V(\sigma) \text{ s.t } B \in \lambda(\sigma)\}$$

I.e. the placeholders appearing on $\mathsf{Reg}^{\text{bot}}_{\mathcal{C},\mathcal{T}}$ vertices in σ. Now we define $\mathcal{A}_{\text{m}}$ to simply ensure we always have $\mathbf{B}(\xi) \subseteq \mathbf{B}_{\text{bot}}(\sigma)$. More precisely, we define $\mathcal{A}_{\text{m}} = (Q, q_0, \delta_{\text{m}}, (\emptyset, Q))$ where

- $Q = \{q_0\}$
- For every $(\sigma, \xi) \in \Sigma_{\text{fr}} \times \Xi$ where $\mathbf{B}(\xi) \subseteq \mathbf{B}_{\text{bot}}(\sigma)$, we have $(q_0, (\sigma, \xi)) \ni (q_0, \ldots, q_0)$
- $U = \{(\emptyset, q_0)\}$

5.3.5 Putting the automata together

Finally, we build $\mathcal{A}_{\mathcal{T},\mathcal{C}}$ as the intersection of $\mathcal{A}'_{\text{emb}}$, $\mathcal{A}'_{\text{alcf}}$, and $\mathcal{A}_{\text{m}}$. Each tree it accepts represents a model of the abstraction of $\mathcal{C}$ w.r.t. $\mathcal{T}$ whose constraint graph can be embedded into $\mathbb{Z}$, yielding the desired reduction of satisfiability to automata emptiness.

Proposition 5.39. *There is a Rabin tree automaton $\mathcal{A}_{\mathcal{T},\mathcal{C}}$ whose state set is bounded by a single exponential in the size of $\mathcal{C}$ and $\mathcal{T}$ and the number of pairs in its Ω is bounded by a polynomial in the size of $\mathcal{C}$ and $\mathcal{T}$, such that $\mathscr{L}(\mathcal{A}_{\mathcal{T},\mathcal{C}}) \neq \emptyset$ if and only if $\mathcal{C}$ is satisfiable w.r.t. $\mathcal{T}$.*

Since emptiness of Rabin tree automata is decidable in time polynomial in Q and exponential in the number of pairs in Ω (Theorem 3.11) we get the main result of this part of the thesis:

Theorem 5.40. *Concept satisfiability w.r.t. general TBoxes in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ is decidable in ExpTime.*

This bound is tight since satisfiability w.r.t. general TBoxes is ExpTime-hard already for plain $\mathcal{ALC}$ (Theorem 3.8).

It is rather surprising that we can add integers to $\mathcal{ALCF}$ for free. Considering how decidability for similar DLs with concrete domains comes at a price, be it reflected in the

domain, in restrictions on the TBox, or restrictions on the paths used, and considering how long this problem has been open despite being singled out as a very desirable extension – one would think that at the very least we would pay with higher computational complexity. The approach we took with this result is also atypical. Usually, the automata approach to showing satisfiability is to express a precise condition for satisfaction of one's formula and then construct a tree automaton that accepts exactly the trees satisfying the condition. Then that automaton accepts exactly the tree models of the formula. In contrast, we expressed a looser condition which is necessary but not sufficient for ensuring satisfaction in the general case, so on its surface, our automaton having a non-empty language would not indicate satisfiability. Our challenge was in showing that our condition is sufficient for regular trees *and* in ensuring that it is verifiable with a small-enough Rabin tree automaton.

CHAPTER 6

Adding int or nat Predicates to Dense Domains

So far, we were only concerned with $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ and the challenges of testing embeddability of graphs into the integers. But as we mentioned before, $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{D})$ was defined for various concrete domains $\mathcal{D}$ [38], including dense numeric domains such as the rationals $\mathbb{Q}$ or the reals $\mathbb{R}$. In such settings, which offer their own advantages for modeling real-world data, it can be useful to have a predicate which enforces that certain registers hold integer or natural number values (for example, the number of children a parent has). The need for these predicates has been previously expressed in the literature [43, 71], but despite being a long sought-after feature, they have not been fully incorporated into existing dense settings, as their introduction would obviously nullify the crucial assumption that the domain is dense. In this chapter, we consider the real numbers counterpart of $\mathcal{Z}_c$ as a representative setting of a dense numeric domain, and show how to add the predicates int and nat to it while maintaining our ExpTime complexity bounds. For this purpose, in Section 6.2 we will adapt our previous notions of constraint graph and frames, and rephrase our embeddability condition to incorporate a new measure of strict length. In Section 6.3 we will adjust our automata constructions to incorporate these changes.

6.1 $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$

We denote by $\mathcal{R}_{c,\mathsf{int}}$ the real numbers $\mathbb{R}$ with the binary relations $\{<,=\}$, equalities with constants $=c$ for $c \in \mathbb{R}$, and the predicate int, which will enforce that a register value is an integer number. We will occasionally write e.g. $x < 0$, $x \leq y$ as shorthand for $(x < y) \wedge (y = 0)$ and $(x < y) \vee (x = y)$. Below, we adapt the definitions from Chapter 4 to this setting in a straightforward manner.

For the definition of constraints (Definition 4.2), we add the atomic constraint $\mathsf{int}(t)$ for a register term t.

For an $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$ interpretation $\mathcal{I}$ (Definition 4.6), we note that the register function β obviously assigns values from $\mathbb{R}$ to the registers, and we add to the semantics (Definition 4.7) of this setting the interpretation of the int predicate as follows. For a tuple $\vec{v} = (v_0, \ldots, v_n)$ of elements of the domain $\Delta^{\mathcal{I}}$,

- $\mathcal{I}, \vec{v} \models \mathsf{int}(S^i x)$ if and only if $\beta(v_i, x) \in \mathbb{Z}$.

Observation 6.1. *Notice that a predicate* nat *for enforcing natural number values with the semantics*

$$\mathcal{I}, \vec{v} \models \mathsf{nat}(S^i x) \text{ if and only if } \beta(v_i, x) \in \mathbb{N}$$

is given to us for free, since the above holds if and only if

$$\mathcal{I}, \vec{v} \models \mathsf{int}(S^i x) \wedge (S^i y = 0) \wedge (S^i y \leq S^i x)$$

6.1.1 Atomic normal form for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$

Unlike the previous domain $\mathcal{Z}_c$ that was closed under negation, there is no obvious way to express a register having a non-integer value using the other relations. This is concerning, since the ability to obtain negation-free equivalents of the given $\mathcal{C}$ and $\mathcal{T}$ is crucial for performing the Atomic Normal Form transformation which both lowers the depth of the constraints to 1 and externalizes the Boolean combinations of the constraints. This is in turn crucial for defining our notions of constraint graphs, frames, and ultimately our automata constructions. Therefore in order to maintain a transformation to an ANF, instead of expressing $\neg\mathsf{int}$ via Boolean combinations of other constraints, we argue that instances of $\neg\mathsf{int}$ in fact do not affect satisfiability and can be *ignored*, and so $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$ enjoys the Atomic Normal Form nevertheless. The argument uses notation defined in other contexts later in the chapter, so we postpone the precise proof and only give a sketch here.

Roughly speaking, given $\mathcal{C}$ and $\mathcal{T}$ which use $\neg\mathsf{int}(x)$, we will define other $\mathcal{C}'$ and $\mathcal{T}'$ where we enforce that x is not equal to:

- any register y for which $\mathsf{int}(y)$ appears in $\mathcal{C}$ or $\mathcal{T}$
- any register z for which $z = c$ where $c \in \mathbb{Z}$ appears in $\mathcal{C}$ or $\mathcal{T}$

Then we argue that there is a model of $\mathcal{C}$ w.r.t. $\mathcal{T}$ if and only if there is a model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$. The idea is that a model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$ which assigns an integer value to x could just as easily have assigned a slightly different value and remained a model.

6.2 Embeddability condition for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$

With the semantics of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$ well defined, we move on to solving the satisfiability problem of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$ concepts w.r.t. a general TBox. This problem is still decoupled

into a satisfiability check of the abstracted $\mathcal{ALCF}$ part and an embeddability check for the constraint graphs. The notion of abstraction (Definition 5.1) readily transfers to this setting, as we simply have a new placeholder in $\mathbf{B}$ for the int predicate, therefore our $\mathcal{ALCF}$ automaton from Subsection 3.4 still suffices. We will refer to constraint graphs of the abstractions of $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$ concepts and TBoxes as $\mathcal{R}_{c,\mathsf{int}}$-constraint graphs in order to be abundantly clear. We present their full definition here for ease of reading, although it only slightly differs from Definition 5.3. Let $\mathcal{C}$ and $\mathcal{T}$ be an $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$ concept and TBox which are in Atomic Normal Form, and denote by $\mathcal{C}_a$ and $\mathcal{T}_a$ their respective abstractions.

Definition 6.2 ($\mathcal{R}_{c,\mathsf{int}}$-constraint graph)**.** *Let $\mathcal{I}_a = (\Delta^{\mathcal{I}_a}, \cdot^{\mathcal{I}_a})$ be a plain tree-shaped interpretation of $\mathcal{C}_a, \mathcal{T}_a$. The* $\mathcal{R}_{c,\mathsf{int}}$-constraint graph *of $\mathcal{I}_a$ is the directed partially labeled graph $\mathcal{G}_{\mathcal{I}_a} = (V, E, \lambda)$ where $V = \Delta^{\mathcal{I}_a} \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ and $\lambda : V \to 2^{\mathbf{B}}$, and the edge relation $E = E_< \cup E_=$ is such that, for every $(v, y), (u, x) \in V$,*

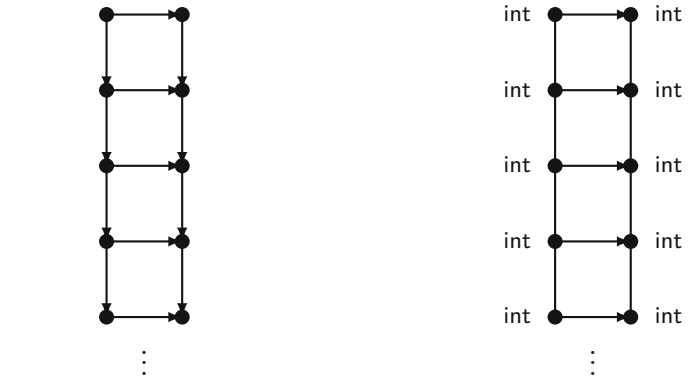
1. *$((v, y), (u, x)) \in E_<$ if and only if either*
   - *$u = v$, $v \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^0 y < S^0 x$,*
   - *u is the parent of $v \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^1 y < S^0 x$, or*
   - *v is the parent of $u \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^0 y < S^1 x$.*
2. *$((v, y), (u, x)) \in E_=$ if and only if*
   - *$u = v$, $v \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^0 y = S^0 x$,*
   - *u is the parent of $v \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^1 y = S^0 x$, or*
   - *v is the parent of $u \in B^{\mathcal{I}_a}$ and B is a placeholder for $S^0 y = S^1 x$.*

In addition, for a placeholder B for $S^0 x = c$ or for $\mathsf{int}(S^0 x)$, we have that $B \in \lambda(u, x)$ if and only if $u \in B^{\mathcal{I}_a}$.

We also adapt the notion of embeddability (Definition 5.4) to this setting in the obvious way:

Definition 6.3 (Embeddability into $\mathcal{R}_{c,\mathsf{int}}$)**.** *We say an $\mathcal{R}_{c,\mathsf{int}}$-constraint graph $\mathcal{G}$ is* embeddable *into $\mathcal{R}_{c,\mathsf{int}}$ if there is a real number assignment $\kappa : \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \to \mathbb{R}$ to the vertices of $\mathcal{G}$ such that for every $(u, x), (v, y) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$*

- *if $((v, y), (u, x)) \in E_<$ then $\kappa(u, x) < \kappa(v, y)$,*
- *if $((v, y), (u, x)) \in E_=$ then $\kappa(u, x) = \kappa(v, y)$,*
- *if $B \in \lambda(u, x)$ is a placeholder for $S^0 x = c$, then $\kappa(u, x) = c$, and*
- *if $B \in \lambda(u, x)$ is a placeholder for $\mathsf{int}(S^0 x)$, then $\kappa(u, x) \in \mathbb{Z}$.*



(a) A graph embeddable into $\mathbb{R}$ despite having a pair with unbounded strict distance

(b) A similar graph that is embeddable into $\mathbb{R}$ despite having infinitely many int-labeled nodes

Figure 6.1: Examples motivating the change in definition of strict length

The embeddability check still boils down to making sure there is no pair of registers with infinitely many int registers between them that must have different values. However, this is no longer tied to the notion of strict length we used so far, as the following examples show. A constraint graph of the form in Figure 6.1(a) with no integer registers is embeddable despite having a pair which violates (★), as it has no strict cycles and $\mathbb{R}$ is dense. Figure 6.1(b) shows that not only can two registers have paths with an unbounded number of strict edges, but they can even have an unbounded number of int registers between them, if we are careful about where the strict edges occur.

The arguments in the proofs of Lemmas 5.15 and 5.16 rely on the notion of strict length corresponding to the number of *different* integer values between two numbers. Therefore in order to utilize those same arguments, we present the more nuanced *strict* int *length* which correctly captures this notion.

Definition 6.4 (Strict int length)**.** *Let p be a finite simple path in an $\mathcal{R}_{c,\mathsf{int}}$-constraint graph. Let $x_1, \ldots, x_n$ be the registers in p which are labeled* int*, in order of appearance in p, i.e. x_i appears before x_j for every $i < j$.*

1. *If $n = 0$, i.e. there are no* int*-labeled registers on p, the strict* int *length of p is 0.*
2. *If $n = 1$, then the strict* int *length of p is 1.*
3. *If $n > 1$, then the strict* int *length of p is $1 + \sum_{i=1}^{n-1} \mathrm{strict}(i)$ where $\mathrm{strict}(i)$ is 1 if there is a strict edge on the path between x_i to x_{i+1}, and 0 otherwise.*

For an infinite path p, we say that p is int *strict if its strict* int *length tends to infinity with the length of its prefixes.*

Going back to the examples in Figure 6.1, we have that the graph on the left has strict int length 0, and the graph on the right has strict int length 2.

Observe that for infinite paths, int strictness can be phrased more simply:

Observation 6.5. *An infinite path is* int *strict if and only if it is both strict in the sense of Definition 5.14 and has infinitely many* int *labels.*

The motivation behind the inductive Definition 6.4 is the inductive structure of the proof of Lemma 5.16, which we will transfer to this setting. But first, we still need to adapt the definitions of tree representations of $\mathcal{R}_{c,\mathsf{int}}$-constraint graphs and of frames. The adaptations capture the same notions conceptually and are not surprising, but some technicalities do need to be handled with regard to the int predicate.

In the $\mathcal{Z}_c$ setting, we had a finite number of choices for the value of a register that lied between two constants, so we only needed to consider the integers between the largest and the smallest integers used in $\mathcal{C}$ and $\mathcal{T}$, as well as two ranges for the values larger than the largest constant and smaller than the smallest constant. However, in this setting, we can no longer simply list all possible values that non-int registers take, even when we explicitly have the constants they are smaller and larger than. Therefore we must make do with bounding intervals, in which we assume register values lie. We will use the notation $(-\infty, c)$ for $\{c' \mid c' \in \mathbb{R}, c' < c\}$ and similarly for (c, ∞).

For example, if $\mathcal{C}$ and $\mathcal{T}$ only use the constants 0 and 0.5, we would have that every register takes a value which either relates to 0 and 0.5 by equality, or relates to the intervals $(-\infty, 0)$, $(0, 0.5)$, and $(0.5, \infty)$ by membership. We make this notion precise and define the $\mathbf{U}$ labels in a way that takes into account possible values when it is an int register and possible interval memberships for non-int registers.

Let c_0 be the smallest constant used in either $\mathcal{C}$ or $\mathcal{T}$ and let c_α be the largest. If no constants were used, set $c_0 = c_\alpha = 0$. Denote by $\mathsf{int}[c_0, c_\alpha]$ the range of *integers* between c_0 and c_α, including c_0 if it is an integer, and likewise for c_α. We will use these labels in the same fashion that we did in the $\mathcal{Z}_c$ setting, in order to assign a concrete integer to the int registers. Additionally, denote by $\mathsf{non\text{-}int}(\mathcal{C}, \mathcal{T})$ the set of non-integer constants used in either $\mathcal{C}$ or $\mathcal{T}$. In order to use these to relate the value of non-int registers as accurately as we can to the constants used in either $\mathcal{C}$ or $\mathcal{T}$, we also define the partition of $\mathbb{R}$ into intervals induced by $\mathbf{C} = \mathsf{int}[c_0, c_\alpha] \cup \mathsf{non\text{-}int}(\mathcal{C}, \mathcal{T})$. Let $c_0, \ldots, c_\alpha$ the elements of $\mathbf{C}$, listed in the usual linear order over $\mathbb{R}$. Obviously, $\mathbb{R}$ excluding the elements of $\mathbf{C}$ can be partitioned into the intervals $(-\infty, c_0), (c_0, c_1), \ldots, (c_{\alpha-1}, c_\alpha), (c_\alpha, \infty)$. Notice that there is indeed no overlap between any of the intervals.

We again define a set of fresh labels $\mathbf{U}$, using the components set up above:

$$\begin{aligned}\mathbf{U} = &\{U_{<c_0}, U_{c_\alpha<}\} \cup \{U_{(c_i,c_{i+1})} \mid i = 0, \ldots, \alpha - 1\} \cup \\ &\{U_c \mid c \in \mathsf{int}[c_0, c_\alpha]\} \cup \{U_c \mid c \in \mathsf{non\text{-}int}(\mathcal{C}, \mathcal{T})\} \cup \{U_{\mathsf{int}}\}\end{aligned}$$

Observe there is an obvious linear order on $\mathbf{U}$, which we will denote $<_{\mathbf{U}}$. We will utilize it to somewhat simplify our later definitions. We redefine the alphabet Σ to be the set of partially $\mathbf{U}$-labeled graphs where the vertex set is either exactly

$$V = \mathsf{Reg}^{\mathrm{top}}_{\mathcal{C},\mathcal{T}} \cup \mathsf{Reg}^{\mathrm{bot}}_{\mathcal{C},\mathcal{T}} \quad \text{or} \quad V = \mathsf{Reg}^{\mathrm{bot}}_{\mathcal{C},\mathcal{T}}$$

with edge set $E = E_< \cup E_=$. Then the tree representations of $\mathcal{R}_{c,\mathsf{int}}$-constraint graphs are defined as in Section 5.2. Notice that only the labels indicating equality with a constant are present in these representations; the labels indicating membership in an interval will come into play in the definition of frames below. This definition is long and verbose, but it simply spells out that the way vertices are labeled and connected is maximal and respects the linear order on $\mathbb{R}$.

Definition 6.6 ($\mathcal{R}_{c,\mathsf{int}}$-Frame)**.** *An $\mathcal{R}_{c,\mathsf{int}}$-frame is a graph in Σ such that:*

1. *there is an edge between every pair of vertices*
2. *there are no strict cycles, i.e. no cycles that include an edge from $E_<$*
3. *equality edges are symmetric, i.e. if $e_=(x, y)$ then also $e_=(y, x)$*
4. *every vertex must have exactly one of the labels in $\mathbf{U} \setminus \{U_{\mathsf{int}}\}$, and*
   - *if a vertex is labeled with a label from $\{U_c \mid c \in \mathsf{int}[c_0, c_\alpha]\}$ then it is also labeled with U_{int}*
   - *if a vertex is labeled with a label from*

     $$\{U_{(c_i,c_{i+1})} \mid i = 0, \ldots, \alpha - 1\} \cup \{U_c \mid c \in \mathsf{non\text{-}int}(\mathcal{C}, \mathcal{T})\}$$

     then it is not labeled with U_{int}
5. *if $e_=(x, y)$ then x and y have the same label from $\mathbf{U}$, and if x and y have the same label from $\{U_c \mid c \in \mathsf{int}[c_0, c_\alpha]\} \cup \{U_c \mid c \in \mathsf{non\text{-}int}(\mathcal{C}, \mathcal{T})\}$, i.e. are labeled with the same constant, then $e_=(x, y)$. Notice that this does not include cases where x and y are labeled with labels that are considered equal in $<_{\mathbf{U}}$ (for example the same interval label), as it does not imply equality between their actual value assignments*
6. *if $e_<(x, y)$ then the label of x is smaller than the label of y in $<_{\mathbf{U}}$*

We denote the alphabet of $\mathcal{R}_{c,\mathsf{int}}$-frames by Σ_{fr}.

The notion of consistent frames (Definition 5.11) readily transfers to this setting.

Finally, we are ready to present the embeddability condition for regular $\mathcal{R}_{c,\mathsf{int}}$-constraint graphs:

($\bigstar_{\mathsf{int}}$) There are no $(u,x),(u,y) \in \Delta \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}}$ in $\mathcal{G}_{\text{fr}}$ for which we have that: there exists an infinite $\mathbf{w} \in [n]^{\omega}$ and

1. an infinite forward path f from (u,x) along $\mathbf{w}$
2. an infinite backward path b from (u,y) along $\mathbf{w}$

such that f or b is int strict, and such that for every $i \in \mathbb{N}$, there is a strict edge from $f(i)$ to $b(i)$.

The condition ($\bigstar_{\mathsf{int}}$) being necessary for the embeddability of $\mathcal{R}_{c,\mathsf{int}}$-constraint graphs follows immediately by Observation 6.5 and the special case where all registers are labeled int.

To see that ($\bigstar_{\mathsf{int}}$) is sufficient for regular $\mathcal{R}_{c,\mathsf{int}}$-constraint graphs, we will show that the notion of strict int length does capture the minimal number of different integer values along a path, and apply the arguments in the proof of Lemma 5.16.

Lemma 6.7. *Let p be an $\mathcal{R}_{c,\mathsf{int}}$-embeddable finite simple path with strict int length n, and let κ be a register value assignment which witnesses its embeddability. Then there are at least n different integer values assigned to the vertices of p.*

Proof. We prove the claim by induction on the strict int length of p. Let $x_1,\dots,x_m$ be the registers in p which are labeled int, in order of appearance in p, i.e. x_i appears before x_j for every $i<j$.

1. If $n=0$ then $m=0$ and the claim follows.
2. If $n=1$, then there is at least one register labeled int and by Definition 6.3, κ must assign an integer value to it. Therefore the claim follows.
3. Now assume the claim holds for n. Let p have strict int length $n+1$ and let p' be the maximal prefix of p that has strict int length exactly n. We specify that p' is maximal since there can be multiple prefixes with this strict int length, even ones that differ in the number of int labeled registers in case they are connected with equality edges.

   Since κ is an embedding of p into $\mathcal{R}_{c,\mathsf{int}}$, so is its restriction to p'. Denote the int registers in p' by $x_1,\dots,x_{m'}$ and note that by the IH, κ assigns n different integer values to $x_1,\dots,x_{m'}$.

   Now we show that since p' is maximal and of strict int length n, there must be a strict edge and then an int labeled register in the remainder of p. Denote by p'' the suffix of p which excludes p'. In order for p to be of strict int length $n+1$, by Definition 6.4 it follows that p'' contains a strict edge which (not necessarily immediately) precedes an int labeled register which we denote x. Therefore we have that $\kappa(x_{m'}) < \kappa(x)$ and that $\kappa(x) \in \mathbb{Z}$, in other words, κ assigns at least one integer value that differs from the n values assigned to $x_1,\dots,x_{m'}$.

□

Corollary 6.8. *If a regular framified $\mathcal{R}_{c,\mathsf{int}}$-constraint graph satisfies ($\bigstar_{\mathsf{int}}$), then it is embeddable into $\mathcal{R}_{c,\mathsf{int}}$.*

Proof. We have from Lemma 5.9 that there are no strict cycles in the graph. Then the result is given with a straightforward application of the proof of Lemma 5.16 using strict int length. □

6.3 Adapting the automata constructions

Here we adapt our automata constructions to the $\mathcal{R}_{c,\mathsf{int}}$ setting. In fact, the only automaton whose adaptation is interesting is the one for checking ($\bigstar_{\mathsf{int}}$); the automaton for checking satisfiability of the $\mathcal{ALCF}$ part readily transfers, and the automata for checking consistency and matching the alphabets require merely syntactic changes which do not change their asymptotic size either.

In order to adapt the automaton $\mathcal{B}$ to its ($\bigstar_{\mathsf{int}}$) counterpart $\mathcal{B}_{\mathsf{int}}$, we use Observation 6.5 and define $\mathcal{B}_{\mathsf{int}}$ so that it guesses whether f or b is strict (in the usual sense of Definition 5.14) and has infinitely many int registers. Since the appearances of the int labels and the strict edges are unrelated, we cannot only have states where both are witnessed at the same time in the U sets. On the other hand, due to the semantics of Rabin tree automata, if we include in the U sets every state where we encounter either a strict edge or an int label, we may accept e.g. infinite paths that are simply strict with no int labels. Therefore we have $\mathcal{B}_{\mathsf{int}}$ remember that it encountered a relevant strict edge until it encounters a vertex labeled with int, and vice versa. This is in contrast to $\mathcal{B}$, whose strictness flag only depended on the current letter.

The acceptance condition for $\mathcal{B}_{\mathsf{int}}$ will include those states where an int label was encountered after a strict edge was encountered, and vice versa. This condition is lossy in a way, as potentially many 'good' states are ignored until the complementing flag is raised, however, it is enough since we require there be *both* infinitely many strict edges *and* infinitely many int labels on the guessed path.

We give a partial description of $\mathcal{B}_{\mathsf{int}}$, and omit the parts that are similar to $\mathcal{B}$.

- The path states in the state set of $\mathcal{B}_{\mathsf{int}}$ have an additional binary flag for indicating whether the automaton just visited a vertex labeled int:

$$Q_p = \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \times \mathsf{Reg}_{\mathcal{C},\mathcal{T}} \times \{f, b\} \times \{0, 1\} \times \{0, 1\}.$$

- As mentioned, $\mathcal{B}_{\mathsf{int}}$ remembers whether it passed a strict edge and whether it extended the path with a vertex labeled int as long as the complementing event has not been seen. For every $i \in [n]$, $h \in \{f, b\}$, $- \in \{0, 1\}$, and $\sim \in \{0, 1\}$, we denote

by $(x, y, h, -, \sim)_i$ the n-tuple containing $(x, y, h, -, \sim)$ for entry i and q_1 for every other entry.

We describe the transition relation for the scenario where $\mathcal{B}_{\mathsf{int}}$ guessed it is f which is strict and has infinitely many int labels, and that the current violating pair of registers is x and y. Exactly as before with $\mathcal{B}$, the automaton $\mathcal{B}_{\mathsf{int}}$ guesses a new pair z and w with which the paths f and b are extended. The conditions of the transition relation of $\mathcal{B}$ apply here too, i.e. we require there be a strict edge from x to y and from z to w, and there not be a strict edge from y to w. The divergence from $\mathcal{B}$ is in the way the flags behave:

– if the current letter σ has z^{bot} labeled with int, then in the next state, the int flag is raised:

$$((x, y, f, 0, 0), \sigma) \ni (z, w, f, 0, 1)_i \qquad ((x, y, f, 1, 0), \sigma) \ni (z, w, f, 1, 1)_i$$
$$((x, y, f, 0, 1), \sigma) \ni (z, w, f, 0, 1)_i \qquad ((x, y, f, 1, 1), \sigma) \ni (z, w, f, 0, 1)_i$$

The only transition of note here is the last one, where we lower the strictness flag and keep the int flag raised, as we are now looking for a strict edge.

– if the current letter σ has z^{bot} not labeled with int, then the only case where the flags change is if both flags were raised:

$$((x, y, f, 0, 0), \sigma) \ni (z, w, f, 0, 0)_i \qquad ((x, y, f, 1, 0), \sigma) \ni (z, w, f, 1, 0)_i$$
$$((x, y, f, 0, 1), \sigma) \ni (z, w, f, 0, 1)_i \qquad ((x, y, f, 1, 1), \sigma) \ni (z, w, f, 0, 0)_i$$

– if the current letter has $e_<(x^{\text{top}}, z^{\text{bot}}) \in \sigma$, then in the next state, the strictness flag is raised:

$$((x, y, f, 0, 0), \sigma) \ni (z, w, f, 1, 0)_i \qquad ((x, y, f, 1, 0), \sigma) \ni (z, w, f, 1, 0)_i$$
$$((x, y, f, 0, 1), \sigma) \ni (z, w, f, 1, 1)_i \qquad ((x, y, f, 1, 1), \sigma) \ni (z, w, f, 1, 0)_i$$

again the only transition of note here is the last one, where we lower the int flag and keep the strictness flag raised as we look for an int labeled vertex.

– if the current letter has $e_=(x^{\text{top}}, z^{\text{bot}}) \in \sigma$, then the only time the flags change is if they were both raised:

$$((x, y, f, 0, 0), \sigma) \ni (z, w, f, 0, 0)_i \qquad ((x, y, f, 1, 0), \sigma) \ni (z, w, f, 1, 0)_i$$
$$((x, y, f, 0, 1), \sigma) \ni (z, w, f, 0, 1)_i \qquad ((x, y, f, 1, 1), \sigma) \ni (z, w, f, 0, 0)_i$$

- Ω has just one pair, where

$$L = \emptyset \qquad \text{and} \qquad U = \{q_1\} \cup \{(x, y, h, 1, 1) \mid x, y \in \mathsf{Reg}_{\mathcal{C},\mathcal{T}}, h \in \{f, b\}\}$$

In order to obtain the automaton $\mathcal{A}_{\text{emb-int}}$ that checks whether ($\bigstar_{\mathsf{int}}$) is satisfied, we complement $\mathcal{B}_{\mathsf{int}}$ and intersect it with the tree consistency automaton.

Proposition 6.9. *The Rabin tree automaton $\mathcal{A}_{\text{emb-int}}$ accepts exactly the consistent trees over the alphabet of $\mathcal{R}_{c,\mathsf{int}}$-frames that satisfy $(\bigstar_{\mathsf{int}})$.*

Observation 6.10. *The number of states of $\mathcal{B}_{\mathsf{int}}$ is polynomial in the size of $\mathcal{C}$ and $\mathcal{T}$, its Ω has one pair, and the alphabet is exponential. Therefore $\mathcal{A}_{\text{emb-int}}$ has a number of states and alphabet exponential in the size of $\mathcal{C}$ and $\mathcal{T}$, and its Ω has a constant number of pairs.*

We can combine all constructions as we did in Chapter 5, and yield the main result of this chapter:

Theorem 6.11. *Concept satisfiability w.r.t. general TBoxes in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$ is decidable in ExpTime.*

And from Observation 6.1 we have:

Corollary 6.12. *Concept satisfiability w.r.t. general TBoxes in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int},\mathsf{nat}})$ is decidable in ExpTime.*

Again, these bounds are tight since satisfiability w.r.t. general TBoxes is ExpTime-hard already for plain $\mathcal{ALC}$ (Theorem 3.8).

6.4 Revisiting the atomic normal form for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$

As we mentioned, the atomic normal form for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$ will not be obtained by eliminating negation from $\neg\mathsf{int}$, but rather by ignoring them as they will not affect satisfiability. Instead of showing exactly this, we will replace negations of int with other constraints which will make our proof simpler. The idea being that instead of explicitly requiring a register to not be an integer, it is enough require it to be different than all the 'mentioned' integers.

Given a concept $\mathcal{C}'$ and a TBox $\mathcal{T}'$ in $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{R}_{c,\mathsf{int}})$, we can convert them and their constraints to NNF in linear time, so we assume they are already in NNF. Recall the set $\mathsf{int}[c_0, c_\alpha]$ containing the range of integers between (possibly including) the smallest constant used, c_0, and the largest constant used, c_α. We describe a procedure for removing the instances of $\neg\mathsf{int}$ from $\mathcal{C}'$ and $\mathcal{T}'$. Let a constraint Θ appear as $\exists P.[\![\Theta]\!]$ or as $\forall P.[\![\Theta]\!]$ for P of depth d, such that Θ includes $\neg\mathsf{int}(S^i x)$ as a sub-constraint. We define the sets of registers which will be required to be different than $S^i x$. Denote by Θ_{int} the set of register terms in Θ (positively) constrained to be integers, that is all $S^j y$ such that $\mathsf{int}(S^j y)$ is a sub-constraint of Θ. Let $\{S^i z_c \mid c \in \mathsf{int}[c_0, c_\alpha]\}$ be such that z_c are fresh register names.

In place of $\neg\mathsf{int}(S^i x)$ in Θ, we write

$$\bigwedge_{S^j y \in \Theta_{\mathsf{int}}} \Big((S^i x < S^j y) \vee (S^j y < S^i x)\Big) \wedge \tag{6.1}$$

$$\bigwedge_{c \in \mathsf{int}[c_0, c_\alpha]} (S^i z_c = c) \wedge \Big((S^i x < S^i z_c) \vee (S^i z_c < S^i x)\Big) \tag{6.2}$$

and obtain a constraint with one fewer instance of $\neg\mathsf{int}$. We repeat for all other sub-constraints and path constraints until all $\neg\mathsf{int}$ constraints are removed.

Now we have some $\tilde{\mathcal{C}}$ and $\tilde{\mathcal{T}}$ which only use int positively and are in NNF, so we proceed with the ANF transformation of Section 4.2, which does not introduce negation. Denote the transformed concept and TBox by $\mathcal{C}$ and $\mathcal{T}$.

It remains to show equisatisfiability:

Lemma 6.13. *$\mathcal{C}'$ is satisfiable w.r.t. $\mathcal{T}'$ if and only if $\mathcal{C}$ is satisfiable w.r.t. $\mathcal{T}$.*

Proof. One direction is easy. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \beta)$ be a model of $\mathcal{C}'$ w.r.t. $\mathcal{T}'$ and let $(u, x) \in \Delta^{\mathcal{I}} \times \mathsf{Reg}$ be constrained with $\neg\mathsf{int}$. Then β assigns a non-integer value to (u, x), and therefore constraints of the form in Equation (6.1) are satisfied by $\mathcal{I}$. Furthermore, β can be expanded to assign c to any register appearing in Equation (6.2). Therefore (an expansion of) $\mathcal{I}$ is a model of $\mathcal{C}$ w.r.t. $\mathcal{T}$.

For the other direction, let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \beta)$ be a model of $\mathcal{C}$ w.r.t. $\mathcal{T}$. We make an inductive argument. Let $(u, x) \in \Delta^{\mathcal{I}} \times \mathsf{Reg}$ be the only register constrained with $\neg\mathsf{int}$ in the original $\mathcal{C}', \mathcal{T}'$. We will show there is $\mathcal{I}' \models_{\mathcal{T}} \mathcal{C}$ such that $\beta'(u, x) \notin \mathbb{Z}$. Assume that $\beta(u, x) \in \mathbb{Z}$. Then due to the constraints of the form in Equation (6.2), we have that $\beta(u, x) \notin \mathsf{int}[c_0, c_\alpha]$, therefore it is either larger than the largest or smaller than the smallest integer in $\mathsf{int}[c_0, c_\alpha]$. Without loss of generality, assume that it is the former. Recall that $\mathcal{C}'$ and $\mathcal{T}'$ are in ANF, so the maximal depth is 1. We will now find a range in $\mathbb{R}$ in which the value of (u, x) may reside, by comparing it to the registers of the logical elements directly related to u. Let $\mathbf{X}$ be the set of the registers of the children of u and the registers of its parent. Let c' be the smallest value assigned by β to a register in $\mathbf{X}$ which is larger than $\beta(u, x)$, and denote $\eta = \frac{c' - \beta(u,x)}{2}$.

Claim 6.14. *Define the register value assignment β' which only differs from β by $\beta'(u, x) = \beta(u, x) + \eta$. Then $\mathcal{I}' = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \beta')$ satisfies $\mathcal{C}'$ w.r.t. $\mathcal{T}'$.*

Proof. Notice that the number of registers which are directly compared to (u, x) is bounded due to $\mathcal{C}$ and $\mathcal{T}$ having depth 1, and they are all in $\mathbf{X}$. Since $\beta'(u, x)$ maintains the same equality and comparison relationship with every element of $\mathbf{X}$ that $\beta(u, x)$ did, all constraints previously satisfied by $\mathcal{I}$ are still satisfied by $\mathcal{I}'$. □

Now we apply this argument inductively and have our proof. □

CHAPTER 7

Other Description Logics with Numeric Values

The description logic $\mathcal{ALCF}^{\mathcal{P}}$ is somewhat non-standard when it comes to DLs with concrete domains, and so in this chapter, we review the typical features and expressive abilities of classical DLs with concrete domains and compare them to the ones offered by $\mathcal{ALCF}^{\mathcal{P}}$. We will see that by using the registers, we can often (but not always) simulate these features in $\mathcal{ALCF}^{\mathcal{P}}$. In addition, we will obtain complexity results (Theorem 7.1) for a closely-related logic by encoding it into $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$.

7.1 Classical concrete domains

We review some typical features of description logics with concrete domains in the classical sense, whose main difference from $\mathcal{ALCF}^{\mathcal{P}}$ is that they compare values on different paths, and that they require these paths of be comprised of functional roles. As there is extensive work on DLs with concrete domains, not all definitions will be given here in full, rather, we will restrict ourselves to a sufficient level of detail in order to maintain focus. We refer to [23] and references therein for full treatments.

For our presentation, we will assume a concrete domain $\mathcal{D}$ in the background, which we will treat as a numeric set $\Delta_{\mathcal{D}}$ with binary or unary relations θ. There are other choices for $\mathcal{D}$, for example concrete domains for temporal [71] and spacial [42] constraints.

In the classical setting, concrete domains are accessible through concrete paths, which are a series of functional roles (dubbed abstract features) followed by a concrete feature which is a (partial) function from logical elements into the concrete domain. The access of the concrete domain is given in a singular way, that is, each concrete path grants access to one element of the concrete domain. More precisely, Let $\mathsf{N_{aF}} \subseteq \mathsf{N_R}$ be an infinite set of *abstract features* such that also $\mathsf{N_R} \setminus \mathsf{N_{aF}}$ is infinite. Let $\mathsf{N_{cF}}$ be an infinite set

of *concrete features*, such that it is disjoint from $\mathsf{N_C}$ and $\mathsf{N_R}$. These are similar to the registers of $\mathcal{ALCF}^{\mathcal{P}}$. A *concrete path* is a sequence $u = f_1 \dots f_k g$, where $f_1, \dots, f_k \in \mathsf{N_{aF}}$ and $g \in \mathsf{N_{cF}}$.

We require that that every concrete feature $g \in \mathsf{N_{cF}}$ is mapped (by some interpretation $\mathcal{I}$) to a partial function from the logical elements $\Delta^{\mathcal{I}}$ to the set $\Delta_{\mathcal{D}}$ of the concrete domain. Then, we have that concrete paths $u = f_1 \dots f_k g$ are interpreted as the composition of their components. That is, for a logical element $e \in \Delta^{\mathcal{I}}$,

$$u^{\mathcal{I}}(e) = g^{\mathcal{I}}(f_k^{\mathcal{I}}(\cdots(f_1^{\mathcal{I}}(e))))$$

Now we can augment $\mathcal{ALC}$ with a concept constructor

$$\exists u_1, u_2.\theta$$

where θ is a binary relation, which is interpreted as the elements from which there is a u_1-path and a u_2-path, whose endpoints satisfy θ:

$$(\exists u_1, u_2.\theta)^{\mathcal{I}} = \{e \in \Delta^{\mathcal{I}} \mid \text{There exist } c_1, c_2 \in \Delta_{\mathcal{D}} \text{ such that } u_1^{\mathcal{I}}(e) = c_1, u_2^{\mathcal{I}}(e) = c_2 \text{ and } \theta(c_1, c_2)\}$$

A similar constructor for unary θ is given in the obvious manner.

It is also common to support the expression of a concrete feature g being *undefined*, using the syntax $g\uparrow$ and the semantics

$$(g\uparrow)^{\mathcal{I}} = \{e \in \Delta^{\mathcal{I}} \mid g^{\mathcal{I}}(e) \text{ is undefined}\}$$

Let us discuss some differences between the classical setting and $\mathcal{ALCF}^{\mathcal{P}}$. In the classical setting, we can compare values that may appear along different paths, whereas $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{D})$ compares values along a single path. Furthermore, $\mathcal{ALCF}^{\mathcal{P}}$ does not support undefined-ness in its syntax. However, we will see below that both of these can be overcome; through the use of additional registers, $\mathcal{ALCF}^{\mathcal{P}}$ can still relate register values on different paths, and undefined-ness can be simulated using a special element of the concrete domain and an additional unary predicate.

One stark difference between $\mathcal{ALCF}^{\mathcal{P}}$ and the classical setting is that in the latter, only functional roles may appear in paths that access the concrete domain. This is an unnatural limitation, as many real-world relations are not functional by default. For example, when modeling organizations one sees non-functional roles both in flat structures (teammates) and hierarchical structures (managers of several teams). Generally, any domain comprised from multiple instances of the same unit, be it persons, processes, or physical objects, will suffer from the inability to aptly model how its pieces relate to each other.

There have been efforts to mitigate this issue, with some success, by accepting higher computational cost and applying other restrictions. In [22] the DL $\mathcal{ALCFP}(\mathcal{D})$ is

introduced (using the notation $\mathcal{ALCP}(\mathcal{D})$), which allows arbitrary roles in its concrete domain concept constructor, and for which pure concept satisfiability is NExpTime-complete. However, satisfiability in the presence of general TBoxes is undecidable for any arithmetic domain, where a concrete domain is said to be *arithmetic* if it contains the natural numbers, supports equality and equality with 0 and 1, and supports addition and multiplication. The logic $\mathbb{Q}$-$\mathcal{SHIQ}$, introduced by Lutz in [43], maintains decidability in the presence of general TBoxes and allows the usual access to the concrete domain as well as non-functional access, but in the latter case, the access is restricted to paths of length 1. In contrast, $\mathcal{ALCF}^{\mathcal{P}}$ allows arbitrary roles to appear on its paths without restriction.

As we mentioned, $\mathcal{ALCFP}(\mathcal{D})$ becomes undecidable in the presence of general TBoxes for any arithmetic domain. Notably, the concrete domain $\mathcal{Z}_c$ is not arithmetic as it does not support addition or multiplication, which raises the question whether there is something to be said about concept satisfiability w.r.t. a general TBox in $\mathcal{ALCFP}(\mathcal{Z}_c)$. Indeed, we will see in Section 7.2 that $\mathcal{ALCFP}(\mathcal{Z}_c)$ can be translated into $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ to obtain novel upper complexity bounds on this problem.

Another point worth discussing is how the ability to express Boolean combinations of constraints is affected by whether the roles leading to the concrete features are functional or not. The syntax of the classical setting of DLs with concrete domains typically does not allow one to explicitly express Boolean combinations of the predicates, but when only *functional* roles appear in the concrete paths, Boolean combinations of the constraints can be expressed 'outside' the paths, using the logical connectives between the concepts.

Say we want to express that an element can reach another element with several concrete features (or registers, as we call them), and we would like to express a Boolean constraint on those registers. For example, an r role leading to an element with registers x, y, z, which satisfy $(x < y) \wedge (x = z)$, depicted in Figure 7.1. This is not supported by the syntax of the classical setting, since it only requires these paths to grant access to one register, and it only allows constraints in the form of a single predicate of the concrete domain. It is not an issue if r is functional, since we can express the above constraints as

$$\exists r, r(x < y) \sqcap \exists r, r(x = z)$$

as r will lead to the same logical element.

However, if r is not required to be functional, the fact that syntactically the Boolean combinations are of *concepts* and not of constraints changes the semantics. In our example, instead of comparing the concrete features of one logical element, it is possible we would be comparing the concrete features of 4 different logical elements. In contrast, $\mathcal{ALCF}^{\mathcal{P}}$ supports Boolean combinations of the constraints inside the role-path concept constructor, and the roles grant access to all the registers of a logical element simultaneously, so to speak.

Finally, we mention a feature of some DLs with concrete domains that cannot be simulated by $\mathcal{ALCF}^{\mathcal{P}}$, namely feature (dis)agreements. This is the ability to express whether

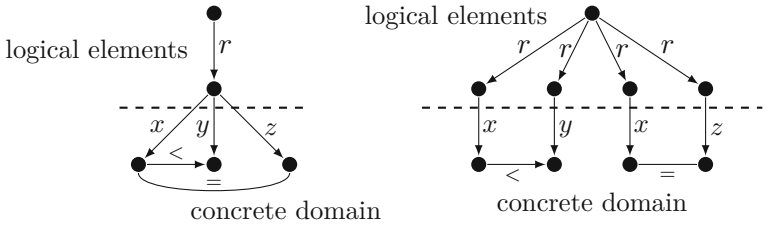


Figure 7.1: Possible interpretations of $\exists r, r(x < y) \sqcap \exists r, r(x = z)$ when r is functional (left) and when r is non-functional (right).

sequences $p_1 = f_1 \dots f_k$ of abstract features (called abstract paths) lead to the same logical element or not. More precisely, for abstract paths p_1 and p_2, we have *feature agreement* with syntax $p_1 \downarrow p_2$ and semantics $\{e \mid \exists e' \text{ s.t. } p_1^{\mathcal{I}}(e) = p_2^{\mathcal{I}}(e) = e'\}$. Feature *disagreement* has the syntax $p_1 \uparrow p_2$ and semantics $\{e \mid \exists e_1, e_2 \text{ s.t. } p_1^{\mathcal{I}}(e) = e_1, p_2^{\mathcal{I}}(e) = e_2 \text{ and } e_1 \neq e_2\}$. Concept satisfiability w.r.t. general TBoxes for $\mathcal{ALC}$ with feature agreements (also denoted $\mathcal{ALCF}$ in the literature) is undecidable [72], although decidability can be regained by disallowing TBoxes [59] or by only allowing acyclic TBoxes [73]. Thanks to the simultaneous access $\mathcal{ALCF}^{\mathcal{P}}$ grants to all the registers, feature disagreement can be simulated by enforcing auxiliary registers to be different and some additional work to overcome the discrepancy between the two-paths-constraints of the classical setting and the single-path-constraints of $\mathcal{ALCF}^{\mathcal{P}}$. We will return to this in the next section.

7.1.1 Supporting undefined register values in $\mathcal{ALCF}^{\mathcal{P}}$

As mentioned above, although $\mathcal{ALCF}^{\mathcal{P}}$ does not offer an "undefined" constructor in its syntax, we can still support expressing that some register values are undefined in our setting. For this, we expand $\mathcal{Z}_c$ to $\mathcal{Z}_{c,\mathsf{und}}$ by adding a fresh element to the integers to obtain $\mathbb{Z} \cup \{\mathsf{u}\}$, and adding a unary predicate und to the predicates of $\mathcal{Z}_c$. Our approach is adapted by redefining frames as follows. The set $\mathbf{U}$ of labels also includes U_{und}, and the first condition in Definition 5.7 is rephrased to be:

1. There is an edge between every pair of vertices which are not labeled U_{und}.

Note that due to condition 5, also every pair of vertices labeled U_{und} is connected (with an equality edge). Clearly, this treatment of undefined register values does not exactly mirror $\uparrow g$, whose semantics would not allow comparisons of undefined register values, whereas our treatment only partially prohibits this; comparisons between registers where one holds a value and the other does not are nonsensical, however, strictly speaking, all registers whose values are undefined are equal to each other. Therefore care must be taken when using U_{und}, and additional restrictions may need to be applied depending on the use case, either in the allowed syntax or using additional TBox axioms. We refer to [45] for more details.

7.2 Simulating the logic $\mathcal{ALCFP}(\mathcal{Z}_c)$ with $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$

The DL $\mathcal{ALCFP}(\mathcal{D})$ was introduced for a general domain $\mathcal{D}$ in [22] using the notation $\mathcal{ALCP}(\mathcal{D})$, and a related DL was studied already in [39]. Here we consider $\mathcal{ALCFP}(\mathcal{Z}_c)$ and show that $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ can simulate it. As previously mentioned, $\mathcal{ALCFP}(\mathcal{Z}_c)$ allows arbitrary roles to participate in paths referring to the concrete domain, similarly to $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$. The bigger hurdle will be that $\mathcal{ALCFP}(\mathcal{Z}_c)$ can compare values on different paths, while $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ can only compare values on the same path.

We briefly recall the definition of $\mathcal{ALCFP}(\mathcal{Z}_c)$, and refer to [40] for details. Since the syntax of $\mathcal{ALCFP}$ does not allow Boolean combinations of constraints, we enrich $\mathcal{Z}_c$ to a richer signature, and explicitly include $\neq, \leq$, and $\geq$. This provides a closer comparison between the logics, and in the case of $\mathcal{ALCF}^{\mathcal{P}}$, it is equivalent to the simpler $\mathcal{Z}_c$ considered so far, since $\mathcal{ALCF}^{\mathcal{P}}$ allows Boolean combinations of constraints. In what follows, we do not repeat the explicit treatment of $x\uparrow$ as it was already previously discussed.

$\mathcal{ALCFP}(\mathcal{Z}_c)$ is the augmentation of $\mathcal{ALCF}$ with:

- $\exists Px. = c$ and $\forall Px. = c$ where $c \in \mathbb{Z}$ and P is a sequence of role names and x a register name, and similarly for $\neq c$. The formulas apply the constraint to the x register of the last element on a P path.
- $\exists P_1x_1, P_2x_2.\theta$ and $\forall P_1x_1, P_2x_2.\theta$ where $\theta \in \{\leq, <, =, \neq, >, \geq\}$ and each P_ix_i is a sequence of role names followed by a register name. The formulas apply the constraint to the x_1 register of the last element on a P_1 path and the x_2 register of the last element on a P_2 path, where both paths start at a common element.

An example which illustrates the difference between $\mathcal{ALCFP}(\mathcal{Z}_c)$ and $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ would be having a friend who is older than a colleague. This concept is expressed naturally in $\mathcal{ALCFP}(\mathcal{Z}_c)$:

$$\exists \text{friend}\, age, \text{colleague}\, age. >$$

It is not immediately clear how to express this $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, since the friend and the colleague are not necessarily reachable from the member element via the same path. We will see how to overcome this using additional registers, as we translate $\mathcal{ALCFP}(\mathcal{Z}_c)$ to $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$. The main idea is that the values of interest at the end of the paths can be propagated up to the shared origin, and compared locally.

- For unary predicates, the translation is straightforward: $\exists Px. = c$ translates to $\exists P.[\![S^{|P|}x = c]\!]$ and $\forall Px. = c$ translates to $\forall P.[\![S^{|P|}x = c]\!]$.
- For existential concepts $\exists P_1x_1, P_2x_2.\theta$, an easy translation is possible by using two fresh register names *copy-*g_1 and *copy-*g_2, which intuitively store at the origin the values at the end of P_1 and P_2. Then the constraint θ can be applied locally to

copy-g_1 and *copy-*g_2. For example, $\exists P_1 x_1, P_2 x_2.\, <$ translates to

$$\begin{aligned} C := \exists P_1.[\![S^0 \textit{copy-}x_1 = S^{|P_1|} x_1]\!] \\ \sqcap \exists P_2.[\![S^0 \textit{copy-}x_2 = S^{|P_2|} x_2]\!] \\ \sqcap \exists \varepsilon.[\![S^0 \textit{copy-}x_1 < S^0 \textit{copy-}x_2]\!] \end{aligned} \tag{7.1}$$

The translations for $\theta \in \{\leq, =, \geq, >\}$ are similar.

- In the cases of $\forall P_1 x_1, P_2 x_2.\theta$, we treat the paths of only functional roles differently from the case where arbitrary roles may occur.

  – If all roles occurring in P_1 and P_2 are functional, then an encoding similar to $\exists P_1 x_1, P_2 x_2.\theta$ can be used, since we are guaranteed to only need to compare numbers at the ends of at most two paths. For example, $\forall P_1 x_1, P_2 x_2.<$ translates to

  $$\neg \exists P_1.\top \sqcup \neg \exists P_2.\top \sqcup C$$

  where C is as in Equation (7.1).

  – If non-functional roles occur in P_1 and P_2, then we may need to compare numbers on several paths, and we may need more sophisticated tricks. For $\theta \in \{\leq, =, \geq, >\}$, this is still possible using just a few registers thanks to these relations being transitive. For example, we can translate $\forall P_1 x_1, P_2 x_2.\, <$ as

  $$\begin{aligned} \neg \exists P_1.\top \sqcup (\exists P_1.[\![S^{|P_1|} x_1 = S^0 \textit{copy-}x_1]\!] \\ \sqcap \forall P_1.[\![S^{|P_1|} x_1 \leq S^0 \textit{copy-}x_1]\!] \\ \sqcap \forall P_2.[\![S^{|P_2|} x_2 > S^0 \textit{copy-}x_1]\!]) \end{aligned}$$

  We are essentially ensuring, via *copy-*x_1, that the largest value of x_1 seen with a P_1 path is smaller than every value of x_2 seen with a P_2 path.

  If θ is $\neq$, our translation cannot avoid explicitly comparing all pairs of values which may be seen with these paths, which requires exponentially many new register names: given $\mathcal{C}$, $\mathcal{T}$ we can ascertain a degree k of some tree model (if any model exists). In this model there would be at most $|P_1|^k$ different values to consider for the satisfaction of the constraint. Slightly abusing notation, we express that the x_1 registers at the end of P_1 paths contain values from a finite set which appears in the fresh register names of the common ancestor, and that this set does not intersect with the set of values of the x_2 registers at the end of P_2 paths:

  $$\begin{aligned} \exists \varepsilon.[\![S^0 x_1 \neq \cdots \neq S^0 x_{|P_1|^k}]\!] \\ \sqcap \forall P_1.[\![S^{|P_1|} x_1 = x_1 \vee \cdots \vee S^{|P_1|} x_1 = x_{|P_1|^k}]\!] \\ \sqcap \forall P_2.[\![S^{|P_2|} x_2 = x_1 \vee \cdots \vee S^{|P_2|} x_2 \neq x_{|P_1|^k}]\!] \end{aligned}$$

This translation allows us to give an upper bound on the complexity of reasoning w.r.t. general TBoxes in the DL $\mathcal{ALCFP}(\mathcal{Z}_c)$, which to the best of our knowledge, has never been provided before. Our upper bounds also apply if we replace the integers by the real numbers, with or without int and nat predicates in the concrete domain.

Theorem 7.1. *Satisfiability w.r.t. general TBoxes in $\mathcal{ALCFP}(\mathcal{Z}_c)$ is decidable in 2ExpTime, and it is ExpTime-complete if there is a constant bound on the length of any path P_1 that contains non-functional roles and occurs in a concept of the form $\forall P_1 x_1, P_2 x_2.\neq$.*

Feature disagreement in $\mathcal{ALCF}^{\mathcal{P}}$ Now that we have seen how additional registers can be used to compare register values along different paths, it is easy to see that feature disagreement can be simulated in $\mathcal{ALCF}^{\mathcal{P}}$ by using a dummy register *is-different* and enforcing its copies at the origin to not be equal, similarly to the definition in Equation (7.1).

This concludes the first part of the thesis, where we established complexity bounds for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$ and related settings. Next, we shift our attention to the two-variable fragment $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ and how it relates to our novel Pebble Intervals Automata.

Part II

Automata for Two-Variable First Order Logic with Two Orders

CHAPTER 8

Pebble-intervals Automata

Now we move on to the second part of the thesis, where we consider a rich decidable fragment of First Order logic with data values and develop an automata model for it.

We start by introducing pebble-intervals automata (PIA) in this chapter. Similarly to classical finite-state automata, PIAs are read-once automata for strings. However, they read the input in varying order. Using a fixed set of pebbles $[m] = \{1, \ldots, m\}$, a PIA reads a position by choosing three pebbles $i, j, k \in [m]$ and non-deterministically placing k on a position p between the positions on which the pebbles i and j were previously places. After demonstrating the computational power of PIAs, we show that their emptiness problem is in PSPACE in general, and NL-complete under some restrictions (Theorem 8.15). We finish the chapter by studying some closure properties of the languages they accept (Theorem 8.21), as well as some non-closure properties (Theorem 8.24).

In Chapters 9 and 10 we will introduce $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$, and show that this it is captured by PIAs in the following sense: for each sentence ψ, there is a PIA whose language coincides with the string projection language of ψ, up to a substitution of the letters. As a corollary, we get an automata-theoretic proof for EXPSPACE membership of finite satisfiability for $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ that was established in [46].

8.1 Pebble-intervals automata

We begin by giving a high-level description of pebble-intervals automata (PIA), which are read-once string automata with finite memory. A PIA has a finite number m of pebbles, which it uses to read positions of its input w that are contained in some interval between previously placed pebbles. More precisely, the automaton begins its computation with no pebbles on w, and uses MOVE transitions to place and replace pebbles on w. In a k-$\mathrm{MOVE}_{i,j}$ transition where $i, j, k \in [m]$, the pebble k (which may or may not have

been previously placed on w) is non-deterministically placed on an unread position in the interval between pebbles i and j (note that we cannot place the pebbles on i or on j, as their placement on w indicates that those positions have already been read). The boundaries of the input can also be used as interval boundaries. For example, a k-MOVE$_{i,\triangleleft}$ transition places pebble k to the right of pebble i. In addition, the moving pebble can be the same as one of the bounding pebbles, for example in a k-MOVE$_{k,j}$ transition, pebble k will be placed somewhere to the right of its current position but still to the left of the position of pebble j.

For convenience, we also define *silent* transitions that move to a new state without moving any pebbles, although these can be removed. An accepting run is one that ends in an accepting state, after having read the entire input string.

We now give the precise definition of PIAs.

Definition 8.1 (Pebble-intervals automata)**.** *A* PIA $\mathcal{A}$ *is a tuple* $(\Sigma, m, Q, q_{\text{init}}, F, \delta)$, *where*

1. Σ *is the (finite) alphabet,*
2. $m \in \mathbb{N}$,
3. Q *is the (finite) set of states,*
4. $q_{\text{init}} \in Q$ *is the initial state,*
5. $F \subseteq Q$ *are the accepting states, and*
6. $\delta \subseteq (Q \times Q) \cup (Q \times \textsc{move}_m \times \Sigma \times Q)$ *is the transition relation, with*

$$\textsc{move}_m = \{k\text{-}\textsc{move}_{i,j} \mid i \in [m] \cup \{\triangleright\}, j \in [m] \cup \{\triangleleft\}, k \in [m], i \neq j\}$$

We sometimes omit m *when it is clear from the context. Transitions in* $Q \times \textsc{move} \times \Sigma \times Q$ *are* MOVE *transitions, and transitions in* $Q \times Q$ *are* silent *transitions. The* size *of* $\mathcal{A}$ *is* $|\mathcal{A}| = |\delta| + |\Sigma| + |Q|$.

We will often want to describe the positions of m pebbles on a string of length n during a run of a PIA, which we do using (m, n)-pebble assignments:

Definition 8.2 (Pebble assignment)**.** *An* (m, n)-pebble assignment, *where* $m, n \in \mathbb{N}$, *is a function* $\tau : [m] \to [n] \cup \{\bot\}$ *such that for each* $1 \leq i < j \leq m$, *either* $\tau(i) \neq \tau(j)$ *or* $\tau(i) = \tau(j) = \bot$. *That is, different pebbles cannot be placed on the same position of the input, and the fact that a pebble* j *is not on the input is reflected by* $\tau(j) = \bot$.

By $\hat{\tau} : [m] \cup \{\triangleright, \triangleleft\} \to \{0\} \cup [n+1]$ *we denote the extension of a* (m, n)*-pebble assignment* τ *to* $\triangleright$ *and* $\triangleleft$, *by additionally setting* $\hat{\tau}(\triangleright) = 0$ *and* $\hat{\tau}(\triangleleft) = n + 1$.

We write $\tau[k \mapsto \ell]$ to denote the assignment obtained from τ by redefining $\tau(k)$ to ℓ, as will happen when a PIA either places a pebble on the input for the first time or moves a previously placed pebble. Note that PIA cannot remove pebbles from the input once they are placed, only move them to another position.

8.1.1 Semantics of PIAs

Let $\mathcal{A} = (\Sigma, m, Q, q_{\text{init}}, F, \delta)$ be a PIA and let $u \in \Sigma^\star$ be a string of length $|u|$.

Definition 8.3 (Configurations)**.** *A* configuration *of $\mathcal{A}$ on a string $u \in \Sigma^\star$ is a triple (q, ρ, N) where*

1. *$q \in Q$ is the current state,*
2. *$\rho : [m] \to [|u|] \cup \{\bot\}$ is an $(m, |u|)$-pebble assignment which is the current pebble assignment of positions to each pebble, and*
3. *$N \subseteq [|u|]$ is the set of already-read positions of u.*

The initial *configuration π_{init} is $(q_{\text{init}}, \rho_\bot, \emptyset)$, where $\rho_\bot$ is the* initial pebble assignment *defined as $\rho_\bot(k) = \bot$ for every $k \in [m]$, i.e. without any pebbles on the input. A configuration (q, ρ, N) is* accepting *if $q \in F$ and all the positions were read, i.e. $N = [|u|]$.*

Next we describe how the automaton moves from one configuration to the other. Let $\pi = (q, \rho, N)$ and $\pi' = (q', \rho', N')$ be configurations of $\mathcal{A}$ on u. We call them consecutive *and write $\pi \overset{t}{\rightsquigarrow} \pi'$ if there exists a transition t in δ such that either*

1. *t is a silent transition of the form (q, q'), $N = N'$, and $\rho = \rho'$; or*
2. *t is a MOVE transition of the form $(q, k\text{-MOVE}_{i,j}, u(\ell), q')$, which places pebble k on an unread position ℓ in the open interval between i and j, and reads the letter $u(\ell) \in \Sigma$. More precisely, we have that*
   - $\hat{\rho}(i) < \ell < \hat{\rho}(j)$
   - $\ell \in [|u|] - N$
   - $\rho' = \rho[k \mapsto \ell]$
   - $N' = N \cup \{\ell\}$

As we mentioned before, the automaton accepts if it is in an accepting state having read all the input. The following definition makes this precise.

Definition 8.4 (Computation and acceptance)**.** *Let $u \in \Sigma^\star$, $\bar{t} = (t_1, \ldots, t_r)$ be a sequence of transitions and $\bar{\pi} = (\pi_0, \ldots, \pi_r)$ a sequence of configurations on u. We call $(\bar{t}, \bar{\pi})$ a* computation *of $\mathcal{A}$ on u if $\pi_0 = \pi_{\text{init}}$ and $\pi_{i-1} \overset{t_i}{\rightsquigarrow} \pi_i$ for every $i \in [r]$. Then we denote $\pi_0 \overset{\bar{t}}{\rightsquigarrow} \pi_r$. For two configurations π and π', we write $\pi \overset{\star}{\rightsquigarrow} \pi'$ if $\pi \overset{\bar{t}}{\rightsquigarrow} \pi'$ for some $\bar{t}$.*

A computation ending in an accepting configuration is accepting*, and* $\mathcal{A}$ accepts *input* u *if an accepting computation of* $\mathcal{A}$ *on* u *exists. We denote by* $\mathscr{L}(\mathcal{A})$ *the language accepted by* $\mathcal{A}$*. A* PI language *is a language accepted by a PIA.*

8.2 Computational power of pebble-intervals automata

Here we explore the languages that PIA can accept. The first easy observation is that PIA accept all regular languages by using one pebble in a unidirectional way, meaning the pebble is always placed to the right or is always placed to the left of its current location. More precisely:

Definition 8.5 (Single-pebble unidirectional PIA)**.** $\mathcal{A} = (\Sigma, 1, Q, q_{\text{init}}, F, \delta)$ *is* unidirectional *if* q_{init} *has no incoming transitions, and the* MOVE *transitions starting at any other state all use* 1-$\mathit{MOVE}_{1,\triangleleft}$ *only.*

Proposition 8.6. *A string language* L *is accepted by a classical non-deterministic finite-state automaton if and only if* L *is accepted by a unidirectional PIA with the same number of states.*

Proof. Notice that for a unidirectional pebble-intervals automaton with one pebble, every accepting computation must read the letters of the input w in order (i.e. first it reads $w(1)$, then $w(2)$, then $w(3)$, ...), since it can only accept when all position have been read.

Let $\mathcal{A}_{FSA} = (\Sigma, Q, q_{init}, F, \Delta)$ be a non-deterministic finite-state automaton where $\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the transition function. We define the transition relation

$$\delta = \{(q, q') \mid q' \in \Delta(q, \varepsilon)\} \cup \{(q, 1\text{-MOVE}_{1,\triangleleft}, \sigma, q') \mid q' \in \Delta(q, \sigma), \sigma \in \Sigma\}$$

and let $\mathcal{A}_{PI} = (\Sigma, 1, Q, q_{init}, F, \delta)$ be the pebble-intervals automaton. We have $\mathscr{L}(\mathcal{A}_{FSA}) = \mathscr{L}(\mathcal{A}_{PI})$. Conversely, we define Δ from δ as follows: for every $q \in Q$ and $\sigma \in \Sigma$, let

$$\Delta(q, \varepsilon) = \{q' \mid (q, q') \in \delta\} \cup \{q' \mid (q, 1\text{-MOVE}_{1,\triangleleft}, \sigma, q') \in \delta\}$$

□

Examples of PIA languages

Next we describe PIA constructions for some non-regular, and not even context-free languages.

Example 8.7. *First we consider the Dyck language* L_{Dyck} *of well-nested brackets over the alphabet containing the two letters* [*and*]*. The strings in* L_{Dyck} *satisfy that the number of opening brackets is equal to the number of closing brackets, and that in every prefix, the number of closing brackets is at most the number of opening brackets. It is*

well known that L_{Dyck} is context-free but not regular (see e.g. [74]). We will fully describe a PIA $\mathcal{A}_{Dyck} = (\Sigma, 1, Q, q_{\text{init}}, F, \delta)$ that accepts L_{Dyck}, as some following examples are straightforward adaptations of it.

Essentially, $\mathcal{A}_{Dyck}$ will move the pebble to some opening bracket on its input, and attempt to match it to a closing bracket to its right. Note that it will not necessarily match the closing bracket to the bracket that opened it.

For ease of understanding, the construction is not minimal. We have that

- Σ *consists of* [*and*]
- $m = 1$
- $Q = \{q_{\text{init}}, q_[, q_]\}$
- *the initial state is* q_{init}
- *the only state in* F *is* $q_]$
- *the transition relation* δ *contains* $(q_{\text{init}}, 1\text{-}MOVE_{\triangleright,\triangleleft}, [, q_[)$, $(q_[, 1\text{-}MOVE_{1,\triangleleft},], q_])$, *and* $(q_], 1\text{-}MOVE_{\triangleright,\triangleleft}, [, q_[)$.

Example 8.8. *The PIA $\mathcal{A}_{Dyck}$ can be easily adapted to accept the language L_{two}, which consists of all strings in which two types of parentheses may occur, such that each type of parentheses is well-nested with respect to itself, but not necessarily with respect to the other type. For example, the string ([ )] is in L_{two}, but ( ] is not. L_{two} is not context-free. The accepting PIA of L_{two} is obtained from $\mathcal{A}_{Dyck}$ by essentially duplicating its state set so that it first consumes all the brackets of one kind and then consumes all brackets of the second kind.*

We suspect that the language $L_{wn\text{-}two}$ of well-nested parentheses of two kinds is not a PI language. As opposed to L_{two}, this language requires that the parentheses are well-nested with respect to both kinds, so for example, we have ([ )] $\in L_{two}$ but ([ )] $\notin L_{wn\text{-}two}$. Therefore it would seem that PI language are incomparable to context-free languages.

Example 8.9. *The* MIX *language, which is the language over $\{a, b, c\}^+$ where each letter occurs the same number of times, is not context-free and is also PI. The accepting PIA repeats a loop for a non-deterministic number of times, where in each iteration it reads one a, one b, and one c that appear anywhere on the input, thus ensuring they occur an equal number of times.*

Example 8.10. *The language $\{a^n\$b^n\#c^n \mid n \geq 0\}$, which is not context-free, is accepted by a PIA with* 3 *pebbles. Pebbles* 1 *and* 2 *read the* \$ *and the* #, *and then the automaton repeats the following a non-deterministic number of times: pebble* 3 *reads an 'a' to the left of pebble* 1, *a 'b' between pebbles* 1, 2, *and a 'c' to the right of pebble* 2.

Example 8.11. *The language* $\{w\$w \mid w \in \Sigma^+\}$ *for* $\Sigma = \{0,1\}$ *is not context-free, and is accepted by a PIA with* 3 *pebbles. Pebble* 1 *reads the* \$*, and pebbles* 2 *and* 3 *are used to read the same letters to the left and to the right of pebble* 1*, such that in an accepting run, pebble* 2 *reads the positions of* w *that is to the left of pebble* 1 *in order, and so does pebble* 3 *to the* w *that is on the right of pebble* 1*. This is achieved by having the automaton repeat the following a non-deterministic number of times once pebble* 1 *is placed on the* \$*: (i) a letter* σ *is non-deterministically chosen, (ii) pebble* 2 *reads* σ *between its current position and pebble* 1*, and (iii) pebble* 3 *reads* σ *to the right of its current position.*

8.3 Emptiness of pebble-intervals automata

Here we show that PIA have a decidable emptiness problem, and discuss its complexity. This problem is solved with a usual search of a sequence of transitions that represent an accepting computation. However, not every sequence of transitions that respects the transition relation alone corresponds to a computation of the automaton. For example, a PIA with 3 pebbles cannot make a 1-$\text{MOVE}_{2,3}$ in its first or second transitions, as the pebbles 2 and 3 would not yet be on the input. We therefore define feasible sequences of transitions, which take into account pebble placements.

We use the term *arrangement* of the pebbles $[m]$ to refer to a linear order $\mathcal{O} = \langle O, R_{\leq} \rangle$ with $O \subseteq [m]$. Each (m,n)-pebble assignment ρ induces a unique arrangement $o(\rho)$, where $O = [m] - \rho^{-1}(\bot)$ and $R_{\leq}(i,j)$ holds if and only if $\rho(i) \leq \rho(j)$.

Definition 8.12 (Feasible sequence of transitions)**.** *Let* $r \geq 0$ *and let* $\bar{t} = (t_1, \ldots, t_r)$ *be a sequence of transitions in* δ*. We say* $\bar{t}$ *is* feasible *if there is a sequence* $\bar{\mathcal{O}} = (\mathcal{O}_0, \ldots, \mathcal{O}_r)$ *of arrangements and a sequence* $\bar{q} = (q_0, \ldots, q_r)$ *of states with* $q_{\text{init}} = q_0$*, such that* $\mathcal{O}_0$ *is empty, and for every* $1 \leq \ell \leq r$*:*

1. *if* t_ℓ *is a silent transition, then* $t_\ell = (q_{\ell-1}, q_\ell)$ *and* $\mathcal{O}_\ell = \mathcal{O}_{\ell-1}$*, and*
2. *if* t_ℓ *is a* MOVE *transition, then there are* $k \in [m]$*,* $i, j \in ([m] - \{k\}) \cup \{\triangleright, \triangleleft\}$*, and* $\sigma \in \Sigma$ *with* $t_\ell = (q_{\ell-1}, k\text{-MOVE}_{i,j}, \sigma, q_\ell)$ *and the arrangement* $\mathcal{O}_\ell = \langle O_\ell, R_{\leq,\ell} \rangle$ *is such that*
   - $O_\ell = O_{\ell-1} \cup \{k\}$*,*
   - $R_{\leq,\ell}(i', j')$ *if and only if* $R_{\leq,\ell-1}(i', j')$ *for all for* $i', j' \in [m] - \{k\}$*,*
   - *if* $i \in [m]$ *then* $R_{\leq,\ell}(i,k)$*, and*
   - *if* $j \in [m]$ *then* $R_{\leq,\ell}(k,j)$*.*

The following lemma shows that the feasible sequences of transitions are exactly the ones corresponding to actual computations of the automaton.

Lemma 8.13. *A sequence* $\bar{t}$ *of transitions is feasible if and only if there is a string* $u \in \Sigma^\star$ *and a sequence of configurations* $\bar{\pi}$ *such that* $(\bar{t}, \bar{\pi})$ *is a computation of* $\mathcal{A}$ *on* u*. Moreover,* $\bar{t}$ *ends in an accepting state if and only if* $(\bar{t}, \bar{\pi})$ *is an accepting computation.*

Proof. Let $(\bar{t}, \bar{\pi})$ be a computation with $\bar{t} = (t_1, \ldots, t_r)$, $\bar{\pi} = (\pi_0, \ldots, \pi_r)$, and with $\pi_h = (q_h, \rho_h, N_h)$ for every $h \in \{0, \ldots, r\}$. Let $\mathcal{O}_0$ be the empty structure and let $\mathcal{O}_h = \langle O_h, R_{\leq,h} \rangle$ where $O_h = [m] - \rho_h^{-1}(\bot)$ and $R_{\leq,h} = \{(i,j) \in O_h^2 \mid \rho_h(i) \leq \rho_h(j)\}$ for every $h \in [r]$. The sequence $\bar{\mathcal{O}} = (\mathcal{O}_0, \ldots, \mathcal{O}_r)$ satisfies the requirements in Definition 8.12, and hence $\bar{t}$ is feasible.

Conversely, assume $\bar{t} = (t_1, \ldots, t_r)$ is feasible and let $\bar{\mathcal{O}}$ and $\bar{q}$ be as guaranteed in Definition 8.12. We prove by induction on r that there is a computation $((t_1, \ldots, t_r), (\pi_0, \ldots, \pi_r))$ on some string u_r of length at most r such that $\mathcal{O}_{r-1} = o(\rho_{r-1})$ and $N_r = [|u_r|]$.

For $r = 0$, $(\bar{t}, (\pi_{\text{init}}))$ is a computation on the empty string.

Assume the induction hypothesis holds for a string u_{r-1} of length at most $r-1$ and $((t_1, \ldots, t_{r-1}), (\pi_0, \ldots, \pi_{r-1}))$. Let $\pi_{r-1} = (q_{r-1}, \rho_{r-1}, N_{r-1})$. If we have a silent transition $t_r = (q_{r-1}, q_r)$, then we have $\pi_r = (q_r, \rho_{r-1}, N_{r-1})$, and $(\bar{t}, (\pi_0, \ldots, \pi_r))$ is a computation on u_{r-1}, with $\mathcal{O}_r = \mathcal{O}_{r-1} = o(\rho_{r-1}) = o(\rho_r)$, and $N_r = N_{r-1} = |u_{r-1}|$.

Otherwise, we have a MOVE transition $t_h = (q_{h-1}, k_h\text{-MOVE}_{i_h,j_h}, \sigma, q_h)$. Let

$$b = \begin{cases} \rho_{h-1}(i_h), & \rho_{h-1}(i_h) \neq \bot \\ 0, & \text{otherwise} \end{cases}$$

Let

$$u_r = u(1) \ldots u(b) \sigma u(b+1) \ldots u(r-1),$$

i.e. u_r is obtained by inserting σ into u_{r-1} immediately after the pebble i_h, or at the start of u_{r-1} if the pebble i_h has not been used in the computation so far. For every $h \in \{0, \ldots, r-1\}$, let $\pi'_h = (q_{r-1}, \rho'_{r-1}, N'_{r-1})$ where ρ'_h is obtained from ρ_h by setting $\rho'_r(i) = \rho_{r-1}(i) + 1$ whenever $\rho_{r-1}(i) \in \{b+1, \ldots, r\}$, and let

$$N_h = [b] \cap N_{r-1} \cup \{\ell + 1 \mid \ell \in N_{r-1}, \ell > b\}.$$

Notice that since $((t_1, \ldots, t_{r-1}), (\pi_0, \ldots, \pi_{r-1}))$ is a computation on u_{r-1}, we have that $((t_1, \ldots, t_{r-1}), (\pi'_0, \ldots, \pi'_{r-1}))$ is a computation on u_r. Let ρ'_r be obtained from ρ'_{r-1} by setting $\rho_r(k_h) = b+1$. Let $N_r = N_{r-1} \cup \{b+1\}$, and let $\pi'_r = (q_r, \rho'_r, N'_r)$. By Definition 8.12 (and whether or not $k_r \in \{i_r, j_r\}$), we have

1. $R_{\leq,r}(i_r, k_r)$ and $R_{\leq,r}(k_r, j_r)$, and hence $R_{\leq,r}(i_r, j_r)$, and
2. $R_{\leq,r-1}(i_r, j_r)$ if and only if $R_{\leq,r}(i_r, j_r)$.

Hence, it holds that $R_{\leq,r-1}(i_r, j_r)$. By the induction hypothesis, $\rho_{r-1}(i_r) < \rho_{r-1}(j_r)$. Therefore, $\pi'_{r-1} \rightsquigarrow^{t_r}_{u_r} \pi'_r$ and $(\bar{t}, (\pi'_0, \ldots, \pi'_r))$ is a computation on u_r. By the definitions of $\bar{\mathcal{O}}$, ρ_r, and $o(\rho_r)$, we have $o(\rho_r) = \mathcal{O}_r$. Finally, since $|N_{r-1}| = |u_{r-1}| = |u_r| - 1$ and $N_r - N_{r-1} = \{b+1\}$, we have $N_r = [|u_r|]$.

□

Now emptiness of a PIA $\mathcal{A}$ amounts to the existence of a feasible sequence of transitions that ends in an accepting state, and the following Lemma puts a bound on the length of these sequences.

Lemma 8.14. *There is a feasible sequence of transitions $\bar{t}$ of length at most $|\mathcal{A}|\cdot 2^{O(m \log m)}$ ending in an accepting state if and only if $\mathscr{L}(\mathcal{A}) \neq \emptyset$.*

Proof. By Lemma 8.13, if $\bar{t}$ is feasible and ends in an accepting state, then there is a sequence of configurations $\bar{\pi}$ such that $(\bar{t}, \bar{\pi})$ is an accepting computation of $\mathcal{A}$ on some string u. Hence $u \in L(\mathcal{A})$.

Conversely, assume that $L(\mathcal{A}) \neq \emptyset$, and let $(\bar{t}, \bar{\pi})$ be an accepting computation of minimal length on any string u. Denote $(\bar{t}, \bar{\pi}) = ((t_1, \ldots, t_r), (\pi_0, \ldots, \pi_r))$. By Lemma 8.13, $\bar{t}$ is feasible and ends in an accepting state. Let $\bar{\mathcal{O}}$ be the sequence described in Definition 8.12 for $\bar{t}$. Now assume for contradiction that there are two distinct $h_1, h_2 \in [r]$ such that $t_{h_1} = t_{h_2}$ and $\mathcal{O}_{h_1} = \mathcal{O}_{h_2}$. Then $(t_1, \ldots, t_{h_1}, t_{h_2+1}, \ldots, t_r)$ is a feasible sequence of transitions ending in an accepting state. Hence, there is a word u_{12} which is accepted by a computation of length $r - (h_2 - h_1)$, in contradiction to the minimality of $(\bar{t}, \bar{\pi})$. Hence, the length of $\bar{t}$ is at most $|\delta| \cdot M$, where M is the number of arrangements $\mathcal{O}$ of m pebbles. We have that $M \leq 2^m \cdot m!$ since there are 2^m ways of choosing a subset $O \subseteq [m]$ as the universe of $\mathcal{O}$, and $|O|! \leq m!$ ways to linearly order the set O. □

Finally, we are ready to solve the emptiness problem of PIAs.

Theorem 8.15 (Emptiness of PIA)**.** *If a PIA $\mathcal{A}$ has $O(\log |\mathcal{A}|)$ pebbles, its emptiness problem is* NL*-complete. In general, the emptiness problem for PIAs is in* PSpace*.*

Proof. Let $\mathcal{A} = (\Sigma, m, Q, q_{\text{init}}, F, \delta)$. We non-deterministically attempt to guess a feasible sequence of transitions $\bar{t}$ which ends at an accepting state along with a sequence $\bar{\mathcal{O}}$ as guaranteed in Definition 8.12. From Lemma 8.14, we know that if such a sequence exists, then there is one whose length is at most $|\mathcal{A}| \cdot 2^{O(m \log m)}$. Note that in any point of the algorithm, we only need to simultaneously keep in memory a counter of the sequence length r, two transitions t_h, t_{h+1} for $h \in [r-1]$ and two arrangements $\mathcal{O}_h, \mathcal{O}_{h+1}$. The space used by the counter is logarithmic in $|\mathcal{A}|$. The size of the representation of a transition is $|Q|^2 + |Q|^2 \cdot |\Sigma| \cdot |\text{MOVE}_m|$, i.e. logarithmic in $|\mathcal{A}|$, and the size of the representation of an arrangement is $O(m^2)$. Therefore we have PSpace in general, and membership in NL when m is logarithmic in $|\mathcal{A}|$. Completeness for the NL case follows from the NL-completeness of the emptiness problem of standard finite state automata and Proposition 8.6.

□

We note the following fact, which will prove useful when we relate PIA to a two-variable fragment of First Order logic with data values. It essentially states that we do not need to construct a PIA in full in order to test its emptiness.

Observation 8.16. *Let $\mathcal{A}$ be a PIA with state set Q and for $q \in Q$, let $\delta_q \subseteq \delta$ be the set of transitions from q. We say δ is* NSPACE$(r(|\mathcal{A}|))$-computable *if there is a non-deterministic Turing machine M_δ that runs in space $r(|\mathcal{A}|)$ whose set of possible outputs on input q is δ_q for $q \in Q$. Observe that Theorem 8.15 holds even if $\mathcal{A}$ is not given explicitly, as long as δ is NSPACE$(\log(|\mathcal{A}|))$-computable.*

8.4 Closure properties of pebble-intervals languages

We show some closure properties of PIA languages, which include closure under standard operations like union and concatenation, and also closure under less known operations such as shuffles, which we define next.

Definition 8.17 (Shuffle)**.** *Let $u, v \in \Sigma^\star$ for a finite alphabet Σ. Let $g : [|u|] \to [|u| + |v|]$ be a strictly monotone function, and let $\bar{g} : [|v|] \to [|u| + |v|]$ be the unique strictly monotone function whose image is disjoint from g. The* g-shuffle *of strings u, v is defined as the following string*

$$u \sqcup\!\sqcup_g v = \{w \mid u = w(g(1)) \dots w(g(|u|)), v = w(\bar{g}(1)) \dots w(\bar{g}(|v|))\}.$$

The shuffle *of u, v is defined as:*

$$u \sqcup\!\sqcup v = \{u \sqcup\!\sqcup_g v \mid g : [|u|] \to [|u| + |v|] \text{ is a strictly monotone function}\}.$$

The shuffle of two languages $L, L' \subseteq \Sigma^\star$ *is defined as $L \sqcup\!\sqcup L' = \bigcup_{u \in L, v \in L'} u \sqcup\!\sqcup v$.*

Example 8.18. *Let $u = abc$ and $v = defg$. Their shuffle $u \sqcup\!\sqcup v$ includes all strings containing each letter a, b, c, d, e, f exactly once such that their positions satisfy $a < b < c$ and $d < e < f < g$. This includes, for example, strings such as abcdefg, adebcfg, and deafbgc.*

Definition 8.19 (Iterated shuffle)**.** *The* iterated shuffle *of a language L is defined as $L^{\sqcup\!\sqcup} = \bigcup_{i \geq 0} L_i$ where $L_0 = L$ and $L_{i+1} = L_i \sqcup\!\sqcup L$.*

Example 8.20. *The language L_{two} from Example 8.8 is the iterated shuffle of the set containing the strings ( ) and [ ].*

Theorem 8.21 (Closure properties of PIA)**.** *The class of pebble-intervals languages is effectively closed under union, concatenation, Kleene-$\star$, shuffle, and iterated shuffle.*

Proof. Let $\mathcal{A}_1 = (\Sigma_1, m_1, Q_1, q_{1,\text{init}}, F_1, \delta_1)$, and $\mathcal{A}_2 = (\Sigma_2, m_2, Q_2, q_{2,\text{init}}, F_2, \delta_2)$ be PIA that accept the languages $\mathscr{L}(\mathcal{A}_1)$ and $\mathscr{L}(\mathcal{A}_2)$, respectively. For simplicity, we assume that the pebble sets of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint. In the following, we describe an automaton $\mathcal{A} = (\Sigma, m, Q, q_{\text{init}}, F, \delta)$ which accepts $\mathscr{L}(\mathcal{A}_1) \cup \mathscr{L}(\mathcal{A}_2)$, $\mathscr{L}(\mathcal{A}_1)\mathscr{L}(\mathcal{A}_2)$, etc. For ease of reading, our construction of $\mathcal{A}$ will not necessarily be minimal. Furthermore, our description is given by specifying an accepting run of $\mathcal{A}$, including exact placements of

the pebbles. For example, we may write that $\mathcal{A}$ places a pebble on the first position of the input. Strictly speaking, such behavior cannot be guaranteed, but we write so in order to clarify the desired behavior of the automaton. This is only assumed in cases where the pebbles move in one direction, so whether there is an accepting run is not affected by this assumption.

Union $\mathscr{L}(\mathcal{A}_1) \cup \mathscr{L}(\mathcal{A}_2)$ is accepted by a pebble-intervals automaton $\mathcal{A}$ which has:

1. $\Sigma = \Sigma_1 \cup \Sigma_2$
2. $m = \max\{m_1, m_2\}$
3. $Q = Q_1 \cup Q_2 \cup \{q_{\text{init}}\}$
4. $F = F_1 \cup F_2$
5. $\delta = \delta_1 \cup \delta_2 \cup \{(q_{\text{init}}, q_{1,\text{init}}), (q_{\text{init}}, q_{2,\text{init}})\}$

On input w, the automaton $\mathcal{A}$ guesses whether $w \in \mathscr{L}(\mathcal{A}_1)$ or $w \in \mathscr{L}(\mathcal{A}_2)$ via the silent transitions from the initial state, and then simulates the corresponding automaton on w.

Concatenation We show that $\mathscr{L}(\mathcal{A}_1)\mathscr{L}(\mathcal{A}_2)$ is accepted by a pebble-intervals automaton $\mathcal{A}$. We describe the automaton simultaneously with an accepting run.

Essentially, $\mathcal{A}$ guesses where the partition of the input is and marks it with a special pebble. Then it simulates the corresponding automaton on each segment of the input. Therefore $\mathcal{A}$ has $m = \max\{m_1, m_2\} + 1$ pebbles.

First we handle the case where one or both of the concatenated strings is the empty string. Given input w, the automaton $\mathcal{A}$ guesses whether w is the empty string, therefore we have $(q_{\text{init}}, q_\varepsilon) \in \delta$, where q_ε is a fresh state. If $\varepsilon \in \mathscr{L}(\mathcal{A}_1) \cap \mathscr{L}(\mathcal{A}_2)$, then $q_\varepsilon \in F$.

If $\varepsilon \in \mathscr{L}(\mathcal{A}_1) \setminus \mathscr{L}(\mathcal{A}_2)$, the automaton guesses whether to simulate $\mathcal{A}_2$ on the entire input or not. Therefore we have $Q_2 \subseteq Q$, and $(q_{\text{init}}, q_{2,\text{init}}) \in \delta$. To accept, we take a fresh state $q_{\mathcal{A}_2} \in F$ and for every $q_2 \in F_2$ we have $(q_2, q_{\mathcal{A}_2}) \in \delta$. The case where $\varepsilon \in \mathscr{L}(\mathcal{A}_2) \setminus \mathscr{L}(\mathcal{A}_1)$ is treated similarly.

If $\mathcal{A}$ interprets the input as a concatenation of two non-empty strings, it places the special pebble m on an arbitrary position ℓ in w, which is treated as the last position of the first segment, i.e. the end of the string on which $\mathcal{A}_1$ will be simulated. Since $\mathcal{A}$ will not be able to place a pebble on position ℓ when simulating $\mathcal{A}_1$, it will need to remember the letter that was read in this position in order to correctly simulate $\mathcal{A}_1$ placing a pebble on it, and $\mathcal{A}$ also needs to remember whether $\mathcal{A}_1$ was already simulated to have read the position or not. Therefore we have $q_1^{\sigma_1} \in Q$ for every $q_1 \in Q_1$ and every $\sigma_1 \in \Sigma_1$, which are states that $\mathcal{A}$ will use to simulate $\mathcal{A}_1$ before the letter is read, and we have states $q_1^{\text{r}} \in Q$ for every $q_1 \in Q_1$ to simulate $\mathcal{A}_1$ after the last position was read ($\mathcal{A}$ does not need to remember the letter at position ℓ once it is read).

The placement of the special pebble takes place with a transition of the form $(q_{\text{init}}, m\text{-MOVE}_{\triangleright,\triangleleft}, \sigma_1, q_1^{\sigma_1}) \in \delta$ for $\sigma_1 \in \Sigma_1$.

We describe the simulation of $\mathcal{A}_1$ in two phases; up to and including the reading of the last position, and after having read the last position. We focus on k-MOVE$_{i,\triangleleft}$ transitions since other transitions are simulated by $\mathcal{A}$ in the obvious way.

Before position ℓ is simulated to have been read, in every $(q_1, k\text{-MOVE}_{i,\triangleleft}, \sigma, q_1')$ transition of $\mathcal{A}_1$, the automaton $\mathcal{A}$ will guess whether $\mathcal{A}_1$ will be simulated to read the last position ℓ or not. If $\mathcal{A}_1$ is guessed to not read the last position, $\mathcal{A}$ will make a $(q_1^{\sigma_1}, k\text{-MOVE}_{i,m}, \sigma, q_1^{\sigma_1'})$ transition (note that $\mathcal{A}$ uses the special pebble m instead of $\triangleleft$).

If $\mathcal{A}$ guesses that $\mathcal{A}_1$ does read the last position ℓ of its input, then $\mathcal{A}$ makes a silent transition (since position ℓ has already been visited) of the form $(q_1^{\sigma_1}, q_{1,\text{r}}')$. Since in the run of $\mathcal{A}_1$, the pebble k is now supposed to be on position ℓ, we need to do another form of bookkeeping in order to ensure correct behavior of $\mathcal{A}$ when performing MOVE transitions involving k. Until the pebble k is used by $\mathcal{A}_1$ to read a letter, any i-MOVE$_{j,k}$ transitions of $\mathcal{A}_1$ (where $i \neq k$) are simulated using i-MOVE$_{j,m}$ in $\mathcal{A}$. The next time k is used by $\mathcal{A}_1$ to read a letter, $\mathcal{A}$ makes a MOVE transition with k as well, while taking care of the boundaries; e.g. if it is a k-MOVE$_{i,k}$ transition in $\mathcal{A}_1$, it will be simulated with a k-MOVE$_{i,m}$ in $\mathcal{A}$. Once the pebble k has been used by $\mathcal{A}_1$ to read a position, there is no danger of mismatching boundaries, and the only bookkeeping needed to simulate $\mathcal{A}_1$ is continuing to replace $\triangleright$ with m in MOVE transitions.

In order for $\mathcal{A}$ to start simulating $\mathcal{A}_2$ on the second segment, we enforce that $\mathcal{A}_1$ accepts the first segment of the input. We have silent transitions $(q_{1,\text{r}}, q_{2,\text{init}}) \in \delta$ for every $q_1 \in F_1$.

The simulation of $\mathcal{A}_2$ on the rest of w is straightforward. We use another set of states $q_2' \in Q$ for every $q_2 \in Q_2$ and note that the only difference between the behaviors is that $\mathcal{A}$ simulates k-MOVE$_{\triangleright,i}$ transitions of $\mathcal{A}_2$ with k-MOVE$_{m,i}$ transitions. Note that during the simulation of $\mathcal{A}_2$, no position up to and including ℓ will be read, therefore if $\mathcal{A}_1$ does not accept the first segment of the input, $\mathcal{A}$ will not accept w. We let $q_2' \in F$ for every $q_2 \in F_2$ and have that $\mathcal{A}$ accepts w if and only if it is a concatenation of a string accepted by $\mathcal{A}_1$ with a string accepted by $\mathcal{A}_2$.

Kleene-$\star$ We show that $\mathscr{L}(\mathcal{A}_1)^\star$ is accepted by a pebble-intervals automaton $\mathcal{A}$. This automaton works similarly to the concatenation case, except it uses two pebbles to enclose the substring it simulates an automaton on, and it non-deterministically chooses when to move on to the next segment of the word. We describe a successful run of $\mathcal{A}$.

We have that $\varepsilon \in \mathscr{L}(\mathcal{A}_1)^\star$, therefore there is a transition $(q_{\text{init}}, q_\varepsilon) \in \delta$ with $q_\varepsilon \in F$.

If $\mathcal{A}$ guesses that the input is not the empty string, it begins to simulate $\mathcal{A}_1$ on contiguous substrings of w in rounds, using special pebbles p, p' to enclose the substrings.

In the first round, $\mathcal{A}$ places pebble p on the beginning of the input (since this is an accepting run) and remembers the letter that was read. Then it places pebble p' somewhere

to the right of pebble p and in addition remembers the letter that was read. Then $\mathcal{A}$ simulates $\mathcal{A}_1$ on the substring between the pebbles, similarly to the concatenation case, by replacing k-MOVE$_{\triangleleft,i}$ transitions with k-MOVE$_{p,i}$ transitions, and k-MOVE$_{i,\triangleleft}$ transitions with k-MOVE$_{i,p'}$ transitions. Similarly to the concatenation case, when $\mathcal{A}$ guesses that $\mathcal{A}_1$ is reading one of the positions of p or p' using a pebble k, it uses a silent transition to simulate the placement of pebble k and adjusts boundaries of MOVE transitions involving pebble k until $\mathcal{A}_1$ moves it again.

The subsequent rounds are similar, except that they begin with $\mathcal{A}$ placing p to the right of p', and the simulation of $\mathcal{A}_1$ on that segment of the input does not involve simulating the first position being read. In addition to replacing k-MOVE$_{\triangleright,i}$ transitions with k-MOVE$_{p,i}$ or k-MOVE$_{p',i}$ transitions (depending on the parity of the round number), $\mathcal{A}$ needs to do bookkeeping to ensure that MOVE transitions of $\mathcal{A}_1$ with boundaries involving pebbles that have not been used in the current round yet are not allowed.

Note that for a run to be successful, in addition to the placement of p in the first round being on the first position of the input, in subsequent rounds the placement of p must be immediately to the right of p'.

Every time $\mathcal{A}_1$ is simulated to be in an accepting state, $\mathcal{A}$ non-deterministically decides whether to finish the simulation and to move to an accepting state, finish the current round and start a new one, or continue the current round.

Shuffle The key observation here is that a string which is a shuffle between a string accepted by $\mathcal{A}_1$ and a string accepted by $\mathcal{A}_2$ can be accepted by independently simulating $\mathcal{A}_1$ and $\mathcal{A}_2$ on it, since there is no bookkeeping required to coordinate $\mathcal{A}_1$ and $\mathcal{A}_2$. Therefore, the automaton $\mathcal{A}$ uses two disjoint sets of pebbles, one of size m_1 for $\mathcal{A}_1$ and one of size m_2 for $\mathcal{A}_2$. Using the first set, $\mathcal{A}$ simulates $\mathcal{A}_1$ on some substring of w, which is not necessarily contiguous, and using the second set, it simulates $\mathcal{A}_2$ on the substring composed of the remaining positions. These simulations do not affect each other, and they can be performed one after the other by adding a transition from the accepting states of $\mathcal{A}_1$ to the initial state of $\mathcal{A}_2$. We have that $\mathcal{A}$ accepts if both simulations accept.

Iterated shuffle This is similar to the case of the shuffle, and each iteration will correspond to a round in the run of $\mathcal{A}$. In short, in each round $\mathcal{A}$ simulates $\mathcal{A}_1$ and reads an arbitrary subset of the input positions, and chooses non-deterministically whether to start a new round. As opposed to the previous case, now there is no bound on the number of strings that were shuffled to produce the input, therefore $\mathcal{A}$ must be able to reuse its pebbles, which requires some bookkeeping.

More precisely, in the first round, $\mathcal{A}_1$ is simulated in a straightforward fashion. Whenever $\mathcal{A}_1$ is simulated to go into an accepting state, $\mathcal{A}$ non-deterministically chooses whether to continue the simulation of $\mathcal{A}_1$, or whether one of the shuffled strings in the iterated shuffle was accepted by $\mathcal{A}_1$ and this is the end of a round.

At the end of a round, $\mathcal{A}$ non-deterministically chooses whether to begin another round or end the simulation and accept. If $\mathcal{A}$ chooses to begin another round, it needs to simulate a run of $\mathcal{A}_1$ which starts with all pebbles off the input. However, since there are some pebbles on the input from previous rounds, care must be taken to ensure that the simulation of $\mathcal{A}_1$ only involves feasible transitions. For example, we would like to avoid a situation where pebbles 1 and 2 are already on the input when a new round begins, and $\mathcal{A}$ places the pebble 3 between 1 and 2. Obviously, the semantics of PIA allow this, however, it is not a proper simulation of $\mathcal{A}_1$ on fresh input since a 3-MOVE$_{1,2}$ is not feasible at that point.

Therefore, at the beginning of a new round, $\mathcal{A}$ enters a state $(q_{1,\text{init}}, \{\triangleright, \triangleleft\})$ where all m_1 pebbles are considered to not be placed on the input. $\mathcal{A}$ keeps track of which pebbles are placed on the input in the simulation by using states of the form (q_1, S) where $q_1 \in Q_1$ and $S \subseteq [m_1] \cup \{\triangleright, \triangleleft\}$. As the simulation of $\mathcal{A}_1$ progresses and $\mathcal{A}_1$ is simulated to place pebbles on its input, $\mathcal{A}$ updates the list S; i.e. after a k-MOVE$_{i,j}$ transition into a state (q_1, S), we will have that $k \in S$. We ensure the simulation does not include non-feasible use of the pebbles by only including k-MOVE$_{i,j}$ transitions from a state (q_1, S) if i and j are both in S. That is, $((q_1, S_1), k\text{-MOVE}_{i,j}, (q_2, S_2)) \in \delta$ implies that $i, j \in S_1$ and $\{k\} \cup S_1 \subseteq S_2$.

□

Combining the fact that PI are closed under iterated shuffle with the fact that all regular languages are PI (Proposition 8.6), we immediately have:

Corollary 8.22. *PIA accept the generalizations* MIX$_k$ *to alphabets* $\{\sigma_1, \ldots, \sigma_k\}$*, as they are iterated shuffles of finite (and therefore regular) languages. Namely,* MIX$_k$ *is the shuffle of the set of strings containing exactly one occurrence of each letter* σ_i*,* $i \in [k]$*.*

We will also mention multidimensional Dyck languages [75], which can be seen as MIX languages with a prefix constraint. The k-dimensional Dyck language Dyck$_k$ is the set of strings w in MIX$_k$ such that in every prefix of w, the number of times σ_{i+1} occurs is at most equal to the number of times σ_i occurs, for $i \in [k-1]$. It is easy to see that Dyck$_k$ is the iterated shuffle of $\{\sigma_1 \cdots \sigma_k\}$.

Corollary 8.23. *PIA accept all multidimensional Dyck languages.*

8.5 Non-closure properties of pebble-intervals languages

Unfortunately, PIA are not closed under some useful operations.

Theorem 8.24. *The class of pebble-intervals languages is not effectively closed under:*

1. *Intersection, even with regular languages, and*

2. Complement

To prove this theorem, we will consider some languages related to Minsky machines [76] and make supporting claims. In the first part, we will show that if PI were effectively closed under intersection, then we could decide the Minsky halting problem, which is undecidable. We will use the constructions in that part of the proof in order to show that PI are also not effectively closed under complement.

Proof of Theorem 8.24 (Part 1.) We show that if pebble-intervals automata were effectively closed under intersection with regular languages, we could decide the Minsky halting problem, which is undecidable. A *Minsky machine* [76] is a sequence of labeled commands

$$\begin{array}{ll} \texttt{0}: & \texttt{comm}_0 \\ \texttt{1}: & \texttt{comm}_1 \\ \vdots & \vdots \\ \texttt{n-1}: & \texttt{comm}_{n-1} \\ \texttt{HALT}: & \texttt{halt} \end{array}$$

where each of the first n commands is either an $\texttt{inc}_\texttt{c}$ command of the form:

```
i:  c:= c+1; goto i+1
```

or a conditional $\texttt{dec}_\texttt{c}\texttt{(j)}$ command of the form:

```
i:  if c=0 then goto j
    else c:=c-1; goto i+1
```

and `c` is one of the registers of the machine.

A *trace* of a Minsky machine M is a sequence $\texttt{t}_1, \ldots, \texttt{t}_m$ of labels where

- $\texttt{t}_1 = \texttt{0}$,
- $\texttt{t}_m = \texttt{HALT}$, and
- for $0 < k < m$,
  - if $\texttt{t}_{k-1} = \texttt{i}$ is an $\texttt{inc}_\texttt{c}$ command, then $\texttt{t}_k = \texttt{i+1}$, and
  - if it is a $\texttt{dec}_\texttt{c}\texttt{(j)}$ command, then either $\texttt{t}_k = \texttt{i+1}$, or $\texttt{t}_k = \texttt{j}$.

Note that the language of traces of a Minsky machine M is regular; since there are finitely many commands $\texttt{1}, \ldots, \texttt{n-1}$ which we can encode in finitely many states. The last case does not constrain the possible next labels based on what the value of a register would have been in an actual computation. Those are the traces we define next.

A trace $\mathtt{t}_1, \ldots, \mathtt{t}_m$ is *feasible* if the following holds. If $\mathtt{t}_{k-1} = \mathtt{i}$ for a $\mathtt{dec_c(j)}$ command, and $\mathtt{j} \neq \mathtt{i{+}1}$, then $\mathtt{t}_k = \mathtt{j}$ if and only if $\mathtt{c} = 0$ at step $k-1$ during the run of M. Note that $\mathtt{c} = 0$ holds exactly at the steps where the number of $\mathtt{inc_c}$ commands is equal to the number of $\mathtt{dec_c}$ commands.

The problem of deciding whether a given Minsky machine M with two registers $\mathtt{c}_0, \mathtt{c}_1$ halts when started with $\mathtt{c}_0 = \mathtt{c}_1 = 0$ is referred to here as the *Minsky halting problem*. It is known that the Minsky halting problem is undecidable [77]. In other words, it is undecidable whether there is a feasible trace of M.

Now fix a Minsky machine M and denote the language of its traces by $\mathtt{T}$.

Let $\Sigma = \{\mathtt{halt}, \mathtt{inc_c}, \mathtt{dec_c}, \mathtt{jmp_c} \mid \mathtt{c} \in \{\mathtt{c}_0, \mathtt{c}_1\}\}$. We define a mapping $h : \mathtt{T} \to \Sigma^\star$ from traces to $\Sigma^\star$ which will also produce a regular language. Let $\mathtt{t}_1, \ldots, \mathtt{t}_m$ be a trace. We first define a mapping f from pairs of labels $(\mathtt{t}_{k-1}, \mathtt{t}_k)$ to Σ. Well, strictly speaking, f will map to subsets of Σ, but this will be only in one case and the subset will be of size 2. For $\mathtt{t}_{k-1} = \mathtt{i}$ where $0 < k < m$:

- If $\mathtt{i}$ is an $\mathtt{inc_c}$ command, then $f(\mathtt{t}_{k-1}, \mathtt{t}_k) = \mathtt{inc_c}$.
- If $\mathtt{i}$ is a $\mathtt{dec_c(j)}$ command with $\mathtt{j} \neq \mathtt{i{+}1}$, then

$$f(\mathtt{t}_{k-1}, \mathtt{t}_k) = \begin{cases} \mathtt{dec_c} & \text{if } \mathtt{t}_k = \mathtt{i{+}1}, \\ \mathtt{jmp_c} & \text{if } \mathtt{t}_k = \mathtt{j} \end{cases}$$

- If $\mathtt{i}$ is a $\mathtt{dec_c(i{+}1)}$ command, then $f(\mathtt{t}_{k-1}, \mathtt{t}_k) \in \{\mathtt{dec_c}, \mathtt{jmp_c}\}$.

Now define $h(\mathtt{t}_1 \ldots \mathtt{t}_m)$ as the (set of) strings $f(\mathtt{t}_1, \mathtt{t}_2) f(\mathtt{t}_2, \mathtt{t}_3) \ldots f(\mathtt{t}_{m-1}, \mathtt{t}_m)\,\mathtt{halt}$.

Denote the language resulting from applying h to all the traces, which is the list of commands executed during a trace, by $\mathtt{Inst}_M = \{h(\mathtt{t}) \mid \mathtt{t} \in \mathtt{T}\}$, and note $\mathtt{Inst}_M$ is regular. This language describes the operations performed during a run of a machine. In order to decide whether there exists a *feasible* trace of M, we need to be able to test if the appearances of the $\mathtt{jmp_c}$ letter are in appropriate positions w.r.t the registers. For this purpose, we next define a pebble-intervals language by shuffling two pebble-intervals languages given by a context-free grammar, which can distinguish between appropriately-placed and inappropriately-placed $\mathtt{jmp_c}$ letters. The intersection of the languages will consist of the feasible traces, meaning it will be non-empty if and only if the given Minsky machine halts.

We now define a language $L_\mathtt{c}$ containing sequences of $\mathtt{inc_c}$ and $\mathtt{dec_c}$ which are well matched, along with $\mathtt{jmp_c}$ letters which are appropriately placed. This will be a PI language which we will intersect later with the language of traces to obtain the language of feasible traces of Minsky machines. We define this languages as follows. $w \in L_\mathtt{c}$ if and only if all the following hold:

1. for every position i of w, if $w(i) = \texttt{jmp}_\texttt{c}$, then there is an equal number of $\texttt{inc}_\texttt{c}$ and $\texttt{dec}_\texttt{c}$ in the prefix containing the first i positions
2. in every prefix of w we have that the number of $\texttt{dec}_\texttt{c}$ appearances is not larger than the number of $\texttt{inc}_\texttt{c}$ appearances.
3. the last position of w carries the letter $\texttt{halt}$

Note that strings in this languages do not necessarily correspond to a trace at all. For example, a Minsky machine which lists ten $\texttt{inc}_\texttt{c}$ commands before the first $\texttt{dec}_\texttt{c}$ command will not have any traces starting with $\texttt{inc}_\texttt{c}\texttt{dec}_\texttt{c}$, although there are strings with this prefix that belong to $L_\texttt{c}$.

Claim 8.25. *The language L_c is a pebble-intervals.*

Proof. Essentially, the PIA simulates an altered version $\mathcal{A}'$ of $\mathcal{A}_{Dyck}$ on intervals of its input, where $\texttt{inc}_\texttt{c}$ plays the role of [and $\texttt{dec}_\texttt{c}$ plays the role of]. We describe an accepting run of the automaton for $L_\texttt{c}$.

1. First, it places a special pebble $p_{\texttt{halt}}$ on the last position, reading $\texttt{halt}$.
2. Next, it guesses whether there are any $\texttt{jmp}$ commands in the input.
   a) If it guesses there are not, it reads $\texttt{inc}_\texttt{c}$ positions a non-deterministic number of times, followed by performing the following a non-deterministic number of times: reading an $\texttt{dec}_\texttt{c}$ position and an $\texttt{inc}_\texttt{c}$ positions to its left.
   b) If it guesses there are $\texttt{jmp}$ commands in the input, the automaton reads the positions between subsequent $\texttt{jmp}$ commands to ensure the $\texttt{dec}$ commands match the $\texttt{inc}$ commands in the following way. The cases where the automaton handles the first and second $\texttt{jmp}$ commands are a bit different.
      - For the first $\texttt{jmp}$ position it reads, the PIA places pebble $p_{\texttt{jmp}}$ anywhere on the input, and simulates $\mathcal{A}_{Dyck}$ on the part of the input to the left of $p_{\texttt{jmp}}$. Once $\mathcal{A}'$ is simulated to have accepted, our automaton non-deterministically either moves to an accepting state, or starts a new iteration where it will read the next $\texttt{jmp}$ and its corresponding $\texttt{inc}$ and $\texttt{dec}$ commands.
      - For the second $\texttt{jmp}$ position it reads, the PIA places pebble $p_{\texttt{jmp}'}$ to the right of $p_{\texttt{jmp}}$, and simulates $\mathcal{A}'$ on the interval between $p_{\texttt{jmp}'}$ and $p_{\texttt{jmp}}$, similarly to the previous item.
      - For any subsequent readings of $\texttt{jmp}$ positions, depending on the parity of the iteration number, our PIA places either $p_{\texttt{jmp}'}$ or $p_{\texttt{jmp}}$ to the right of the other one, and simulates $\mathcal{A}'$ on the interval between them.
3. Once all positions of the input are read, our PIA moves to an accepting state.

□

We resume the proof of Theorem 8.24. Consider the languages $L_{\mathtt{c}_0}$ and $L_{\mathtt{c}_1}$, and define $\mathtt{L}$ as the shuffle between $L_{\mathtt{c}_0}$ and $L_{\mathtt{c}_1}$, and note that by Claim 8.25 and Theorem 8.21, it is a pebble-intervals language.

Now observe that intersecting $\mathtt{L}$ with $\mathtt{Inst}_M$ would result in exactly the feasible traces, and assume for contradiction that pebble-intervals languages are effectively closed under intersection with regular languages. We describe a procedure for deciding the Minsky halting problem. Given a Minsky machine M, generate the automaton $\mathcal{A}_{inst}$ for the regular language $\mathtt{Inst}_M$ (Proposition 8.6). Using the assumed effective procedure for intersection with regular languages, now generate an automaton $\mathcal{A}_M$ for

$$L_M = \mathtt{L} \cap \mathtt{Inst}_M.$$

To decide the Minsky halting problem for M, test $\mathcal{A}_M$ for emptiness.

Since emptiness of pebble-intervals automata is decidable, we contradict undecidability of the Minsky halting problem, so we conclude that pebble-intervals languages are not effectively closed under intersection, even with regular languages.

(Part 2.) We show we can build an automaton for $(L_M)^c$ and conclude that if we could effectively build a complement automaton for any pebble-intervals language, then in particular we could build one for $((L_M)^c)^c = L_M$, which we could test for emptiness and again solve the Minsky halting problem.

Since $L_M = \mathtt{L} \cap \mathtt{Inst}_M$, we have that $(L_M)^c = \mathtt{L}^c \cup (\mathtt{Inst}_M)^c$. Note that if $\mathtt{L}^c$ and $(\mathtt{Inst}_M)^c$ are PI languages, then by Theorem 8.21, their union $(L_M)^c$ is a pebble-intervals language, so we separately show that $\mathtt{L}^c$ is a pebble-intervals languages and that so is $(\mathtt{Inst}_M)^c$. Since $(\mathtt{Inst}_M)^c$ is regular, it is also a pebble-intervals language by Proposition 8.6, so all that remains is to show that $\mathtt{L}^c$ is accepted by a pebble-intervals automaton, and we will have our proof. Recall that $\mathtt{L}$ is the shuffle between $L_{\mathtt{c}_0}$ and $L_{\mathtt{c}_1}$.

Claim 8.26. *$\mathtt{L}^c$ is a pebble-intervals language.*

Proof. Note that for every $w \in \mathtt{L}^c$, it holds that there is a register $\mathtt{c}$ and a prefix of w ending with $\mathtt{jmp}_\mathtt{c}$ such that the number of $\mathtt{inc}_\mathtt{c}$ and $\mathtt{dec}_\mathtt{c}$ commands is not equal. The PIA will operate as follows:

1. It will guess the register $\mathtt{c}$, and place a pebble p on the end of the prefix for which this holds, where the $\mathtt{jmp}_\mathtt{c}$ command is.
2. Next it will read all other $\mathtt{jmp}$ positions to the left of pebble p. This is done since PIAs need to read all of their input in order to accept.
3. Next the PIA will guess whether it is the number of $\mathtt{inc}_\mathtt{c}$ commands or the number of $\mathtt{dec}_\mathtt{c}$ commands that is larger, and verify this by doing the following:

a) It will first read positions with the more frequent letter which are to the left of pebble p some non-deterministic number of times.

b) Then switch to a loop that reads $\texttt{dec}_\texttt{c}$ followed by a $\texttt{inc}_\texttt{c}$ to its left, all to the left of pebble p. This loop is repeated a non-deterministic number of times.

4. now all that remains is to read the rest of the input, so the automaton arbitrarily read positions to the right of pebble p and moves to an accepting state.

The automaton accepts a string if and only if it has a violating prefix as described above. □

Corollary 8.27. *The universality and inclusion problems for pebble-intervals automata are undecidable.*

Proof. We have seen in the proof above that we can effectively build an automaton $\mathcal{A}$ for $(L_M)^c$. We have that $\mathcal{A}$ has a universal language if and only if $L_M = \emptyset$. Thus if we could test for universality, we could again solve the Minsky halting problem.

Since universality easily reduces to inclusion, undecidability of the inclusion problem follows. □

Finite jumping automata We will now briefly touch on the automata model that is probably closest to our PIAs in terms of behavior, finite jumping automata [78]. A jumping automaton is essentially a PIA with only one pebble that does not specify the interval in which each transition places the pebble, but instead it puts it on an arbitrary unvisited position. That is, only reads the input using 1-$\text{MOVE}_{\triangleright,\triangleleft}$ transitions. This allows us to transfer some results on jumping finite automata to PIAs. For a language L, we denote by $\text{perm}(L)$ its permutation language, that is the set of permutations of $u \in L$. In [79], it is showed that for every regular or context-free language L, the permutation language $\text{perm}(L)$ is accepted by a finite jumping automaton.

Corollary 8.28. *Let L be either regular or context-free. Then its permutation language* $\text{perm}(L)$ *is a PI language.*

Note however, that PIAs are a strict generalization of jumping finite automata, since there are regular languages such as $\{a\}^\star\{b\}^\star$ that jumping automata do not accept.

Discussion We have introduced PIAs and seen several examples of languages they accept. Notably, they accept families of inherently context-sensitive languages, but the context-free language of well-nested parentheses of two kinds $L_{wn\text{-}two}$ is suspected to not be a PI language. Not surprisingly, PI languages are a proper subset of context-sensitive languages; containment is given by the fact that any PIA can be simulated by a Turing machine using linear space. Inequality is given by the fact that context-sensitive grammars

have an undecidable emptiness problem, while PIAs have an emptiness problem that is in PSPACE in general.

Our investigation of their closure properties revealed that PI languages are closed under some lesser known operations such as shuffles, and our proofs of their non-closure properties contained constructions that also allowed us to show that PIA have undecidable inclusion and universality problems.

The intuition gained in this chapter about PIAs will help us in Chapter 10, as we will be making several transformations of structures to take us from the two-dimensional objects $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ is suited for to the strings PIAs operate on. In order to make those transformations, we will use a normal form for $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ that will break down satisfaction of general formulas into satisfaction of smaller atomic formulas.

CHAPTER 9

Two-variable First Order Logic with Data

In this chapter we provide the necessary background on the two-variable fragment of First Order Logic (FO^2), and discuss extensions thereof that model data. We present the variant $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ along with the corresponding notion of data words, as well as definable data languages and their projections to strings. We finish the chapter with a normal form theorem (Theorem 9.13) that will be used in the next chapter to show how pebble-intervals automata relate to $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$.

Structures All the structures and vocabularies are finite, and we typically denote a structure with a calligraphic font $\mathcal{A}$, its universe by A, and its size by $|A|$. We typically denote vocabularies by voc with a subscript to indicate their relations.

9.1 Two-variable fragment of first order logic and the satisfiability problem

Definition 9.1 (FO^2)**.** *The two-variable fragment of First Order logic (*FO*), denoted* FO^2*, is the restriction of* FO *to formulas that use at most two variables* x *and* y*.*

Example 9.2. *Let* R *be a binary relation symbol. Obviously, the formula*

$$\forall x \forall y \left(R(x,y) \vee R(y,x)\right)$$

is in FO^2*, as is only uses two variables. The formula*

$$\forall x \left(\exists y R(x,y)\right) \vee \left(\exists z R(z,x)\right)$$

uses three variables, but it is equivalent to $\forall x \exists y (R(x,y) \vee R(y,x))$ *which only uses two.*

However, the formula

$$\forall x \forall y \forall z \, (R(x,y) \wedge R(y,z)) \rightarrow R(x,z)$$

which expresses that R is transitive, has no equivalent in FO^2.

The (finite) satisfiability problem of a logic $\mathcal{L}$ is: given a sentence φ in $\mathcal{L}$, does it have a (finite) model? Mortimer first showed that FO^2 has a decidable satisfiability problem [80], by showing that any satisfiable FO^2 sentence φ has a model of size at most double exponential in $|\varphi|$, yielding a nondeterministic double exponential upper bound for the problem. This bound was later improved by Grädel, Kolaitis, and Vardi [81], who showed there exist exponential sized models, thus matching the NExpTime-hardness for FO^2 by Fürer [82].

An important ingredient in these decidability proofs for FO^2 is the Scott Normal Form, which allows one to check satisfaction of the formula using types. Types are essentially maximal consistent sets of atomic and negated atomic formulas, and loosely speaking, all information necessary for determining satisfaction of an FO^2 formula in a structure can be specified using the types of their elements.

Theorem 9.3 (Scott Normal Form [83])**.** *Let ψ be a sentence in* FO^2 *over a vocabulary* voc*. There is a sentence φ of the form*

$$\varphi = \forall x \forall y \, \chi(x,y) \wedge \bigwedge_{b=1}^{B} \forall x \exists y \, \chi_b(x,y)$$

with quantifier free χ and χ_b, such that φ is satisfiable if and only if ψ is satisfiable, and every $\mathcal{A} \models \psi$ has a unique expansion $\mathcal{B} \models \varphi$. Furthermore, φ is of length linear in that of ψ, and is over $\mathrm{voc} \cup \mathrm{voc}_{\mathrm{SNF}}$ *where* $\mathrm{voc}_{\mathrm{SNF}}$ *contains fresh unary relations.*

9.2 FO^2 with order relations

FO^2 has the substantial drawback that it cannot express transitivity [84]. The argument in [84] is that the sentence

$$\forall x \neg R(x,x) \wedge \forall x \exists y R(x,y)$$

where R is transitive becomes an infinity axiom, whereas FO^2 has the finite model property. By the same argument we conclude that FO^2 cannot express that a relation is a linear order, preorder, or an equivalence relation, which are natural candidates for modeling (possibly ordered) data from infinite domains. For example, in a program verification setting involving variables that hold integer values. Nonetheless, satisfiability of FO^2 with these relations can be investigated by only considering structures where certain relations are interpreted in the desired way.

For example, the formula

$$\exists x \exists y (x \neq y) \wedge R(x, y) \wedge R(y, x)$$

is satisfiable in general, however, in models where R is interpreted as a linear order, it becomes unsatisfiable.

There is extensive literature on the (finite) satisfiability of FO^2 with special relations such as orders, equivalence relations, and more generally, transitive relations. The classification of the satisfiability problem of FO^2 with transitive relations is nearly complete, as it is established that satisfiability of FO^2 with two transitive relations is undecidable both in the general and the finite case [85], and decidability of FO^2 extended with a single transitive relation is decidable in both cases [86, 87]. For linear orders, both versions of the satisfiability problem with a single linear order are decidable [24], as well as the finite version with two linear orders [88]. Undecidability follows when three linear orders are present [89].

9.2.1 Data words and their automata models

The efforts to develop logics applicable to property verification in a setting involving data from infinite domains lead to the study of data words. These are (typically finite) strings that in addition to holding letters from a finite alphabet in each position, also hold a data value from an infinite alphabet. Intuitively, we can think of the data as a number attached to each position, although generally the data values may range over any infinite alphabet. There are several ways to model data words and one's access to their data values and positions, each with its own tradeoffs, as FO^2 easily becomes undecidable once order relations are in the mix. A linear order can be used to model strings, where a finite number of unary predicates act as an alphabet. The additional data value that each position carries can be modeled using a second special relation such as an equivalence relation or a preorder. Next, we define data words using a linear order, denoted $\leq_1$, for the positions and a preorder, denoted $\precsim_2$, along with its induced successor relation S_2 to model the data, and motivate this choice.

Definition 9.4 (Preorder)**.** *A total* preorder *relation $\precsim_2$ is a transitive total relation which can be seen as an equivalence relation whose equivalence classes are linearly ordered. We write $x \sim_2 y$ as shorthand for $(x \precsim_2 y) \wedge (y \precsim_2 x)$. The* induced successor relation *S_2 of a total preorder $\precsim_2$ is defined such that for $u \precsim_2 v$, we have $S(u, v)$ if there is no element w such that $u \precsim_2 w \precsim_2 v$*

Preorders are an attractive option for modeling realistic numeric data, since they allow comparisons in terms of which value is larger than which (as opposed to equivalence relations, which only allow equality testing), and since they also allow multiple elements to hold the same data value (as opposed to linear orders, which enforce every position to carry a different data value).

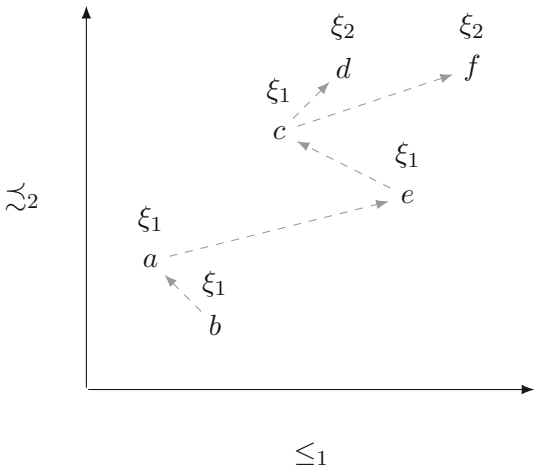


Figure 9.1: The data word $\mathcal{D}$ from Example 9.6.

We denote the extension of FO^2 with a linear order and a preorder and its induced successor relation by $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$, and for a finite alphabet Σ, we denote the *vocabulary* of data words $\langle \leq_1, \precsim_2, S_2, \sigma : \sigma \in \Sigma \rangle$ by $\mathrm{voc}_{\mathrm{DW}}(\Sigma)$.

Definition 9.5 (Data word, data value)**.** *A* data word *over Σ is a finite $\mathrm{voc}_{\mathrm{DW}}(\Sigma)$-structure $\mathcal{D}$ with universe D where σ for $\sigma \in \Sigma$ are interpreted as unary relations that partition D. I.e. each position carries one letter from Σ. The* data value *of an element $d \in D$ is the number $\mathrm{value}_{\mathcal{D}}(d)$ of equivalence classes E of $\sim_2$ whose elements $d' \in E$ satisfy $d' \precsim_2 d$. I.e. what position is its equivalence class in, in their linear order induced by $\precsim_2$. We define $\mathrm{maxval}_{\mathcal{D}} = \max_{d \in D} \mathrm{value}_{\mathcal{D}}(d)$, the largest data value in $\mathcal{D}$.*

We often use $\mathcal{D}$, $\mathcal{D}'$, etc. to denote data words. We denote the empty word by $\emptyset_{\mathrm{DW}(\Sigma)}$, all data words over Σ by $\mathrm{DW}(\Sigma)$, and we call a class of data words a *data language*.

Example 9.6. *Figure 9.1 displays a data word $\mathcal{D}$ over the alphabet $\{\xi_1, \xi_2\}$ whose universe is $D = \{a, b, c, d, e, f\}$, the order interpretations are*

$$a <_1 b <_1 c <_1 d <_1 e <_1 f \qquad \text{and} \qquad b \prec_2 a \prec_2 e \prec_2 c \prec_2 d \sim_2 f,$$

the interpretation of ξ_1 is $\{a, b, c, e\}$, and the interpretation of ξ_2 is $\{d, f\}$. Note e.g. that $\mathcal{D} \models S_2(a, e)$ and $\mathcal{D} \models \neg S_2(b, e) \wedge (b \precsim_2 e)$.

As we mentioned, there are various ways of strings over infinite alphabets, and using FO^2 with a linear order and a preorder is but one of them. Such languages can also studied using automata-theoretic means, and we step back to give a brief survey of the literature. For the moment, we think of data words as strings with a data value attached to each position, and not necessarily as defined in Definition 9.5. Most of the automata models used in that context run on data words, as opposed to the Pebble-intervals automata from

the previous chapter, which run on strings. Register automata, [90, 91, 92], are finite-state machines on data words which use registers to compare whether data values are equal, but do not compare their sizes. The projection languages of register automata are regular, while pebble-interval languages are generally not regular. Pebble automata [91] are automata on data words which use pebbles in a stack discipline to test for equality of data values, and data automata [27, 26, 14] extend register automata. They were introduced to prove the decidability of satisfiability of FO^2 on words with a linear order and a successor relation, with data values which can be compared in terms of equality or inequality. Their projection languages are accepted by multicounter automata, which are finite automata on strings extended with counters, that are equivalent to Vector Addition Systems or Petri Nets [93]. Class Memory Automata [94] have the same expressive power as data automata. Variable Finite Automata [95] extend finite state automata with variables ranging over an infinite alphabet. Their expressive power is incomparable to register automata. Many works have studied these automata models and their variations, we refer to [96] and [97, Chapter 4] for surveys.

9.3 String projections of data words

Our ultimate goal in this part of the thesis is to show the connection between PIAs and languages definable in $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$, for which a tight ExpSpace complexity bound for the satisfiability problem was already provided by Schwentick and Zeume in [46, 98]. We start our discussion with string projections of data words, since PIAs run on strings.

In the sequel, we will make arguments regarding logical implications between formulas that are interpreted over data words. For this, it is convenient to define some notation. Let φ_1, φ_2 be $\mathrm{FO}^2(\mathrm{voc}_{\mathrm{DW}}(\Sigma))$ formulas. We write $\varphi_1 \models_{\mathrm{DW}(\Sigma)} \varphi_2$ when $\mathcal{D} \models \varphi_1$ implies $\mathcal{D} \models \varphi_2$ for every data word $\mathcal{D} \in \mathrm{DW}(\Sigma)$. Logical equivalence $\equiv_{\mathrm{DW}(\Sigma)}$ is defined analogously. When the alphabet is clear from context, we omit Σ and write e.g. $\varphi_1 \models_{\mathrm{DW}} \varphi_2$.

Definition 9.7 (String projection of a data word and the embedding $\mathsf{Emb}_{\mathrm{string},\mathcal{D}}$)**.** *Let $\mathcal{D} \in \mathrm{DW}(\Sigma)$ be a data word of size $|D| \geq 0$. The* string projection *of $\mathcal{D}$, denoted* $\mathrm{string}(\mathcal{D})$*, is the string w of length $|w| = |D|$ where for every position $\ell \in [|w|]$, $w(\ell) = \sigma$ if and only if $\mathcal{D} \models \sigma(d)$ where d is the unique element of $\mathcal{D}$ such that $\ell = |\{d' \in D \mid \mathcal{D} \models d' \leq_1 d\}|$.*

The embedding induced by the string projection, $\mathsf{Emb}_{\mathrm{string},\mathcal{D}}$, is an injective function (indeed a bijection) from $[|w|]$ to D which preserves the order $\leq_1$ of $\mathcal{D}$ in terms of the order of positions in the projection. That is, for any $\ell, \ell' \in [|w|]$ such that $\ell \leq \ell'$, we have that $\mathsf{Emb}_{\mathrm{string},\mathcal{D}}(\ell) \leq_1 \mathsf{Emb}_{\mathrm{string},\mathcal{D}}(\ell')$.

Note that the projection of the empty structure $\emptyset_{\mathrm{voc}_{\mathrm{DW}}(\Sigma)}$ is the empty string ε, and only $\emptyset_{\mathrm{voc}_{\mathrm{DW}}(\Sigma)}$ has ε as its string projection. Despite their somewhat technical definition, string projections of data words are what you would expect; the structure one would obtain from removing the preorder relation from the data word.

Example 9.8. *The string projection of $\mathcal{D}$ from Example 9.6 is*

$$\text{string}(\mathcal{D}) = \xi_1\xi_1\xi_1\xi_2\xi_1\xi_2$$

and $\mathsf{Emb}_{\text{string},\mathcal{D}} : [6] \to D$ *is given by:*

$$\begin{array}{ll} \mathsf{Emb}_{\text{string},\mathcal{D}}(1) = a & \mathsf{Emb}_{\text{string},\mathcal{D}}(4) = d \\ \mathsf{Emb}_{\text{string},\mathcal{D}}(2) = b & \mathsf{Emb}_{\text{string},\mathcal{D}}(5) = e \\ \mathsf{Emb}_{\text{string},\mathcal{D}}(3) = c & \mathsf{Emb}_{\text{string},\mathcal{D}}(6) = f \end{array}$$

Next we can define projection languages:

Definition 9.9 (Projection language)**.** *Let Δ be a data language. The* projection language *of Δ is the language containing the string projections of the data words in Δ:*

$$\mathscr{L}_{\text{str}}(\Delta) = \{w \mid w = \text{string}(\mathcal{D}) \text{ for some } \mathcal{D} \in \Delta\}.$$

We will be interested in (projections of) data languages definable in $\text{FO}^2(\leq_1, \precsim_2, S_2)$, so for a formula ψ which defines Δ, we write $\mathscr{L}_{\text{str}}(\psi)$ for $\mathscr{L}_{\text{str}}(\Delta)$.

Example 9.10. *We can define a data language whose projection language is the Dyck L_{Dyck} of well-nested brackets over the alphabet containing $\sigma_[$ and $\sigma_]$. This is done by enforcing the preorder to have exactly one opening bracket and one closing bracket in each equivalence class, and by enforcing the linear order to be that an opening bracket is always smaller than the closing bracket in its equivalence class. We describe the formulas for each component.*

- *There is at most one opening bracket and at most one closing bracket in an equivalence class:*

$$\forall x \forall y \left((\sigma_[(x) \wedge \sigma_[(y) \wedge x \sim_2 y) \to x = y\right) \wedge \left((\sigma_](x) \wedge \sigma_](y) \wedge x \sim_2 y) \to x = y\right)$$

- *Every opening bracket has a closing bracket in its equivalence class, and vice versa:*

$$\forall x \left(\sigma_[(x) \to (\exists y\, \sigma_](y) \wedge (x \sim_2 y))\right) \wedge \left(\sigma_](x) \to (\exists y\, \sigma_[(y) \wedge (x \sim_2 y))\right)$$

- *The opening bracket in an equivalence class appears before the closing bracket in the equivalence class:*

$$\forall x \forall y (\sigma_[(x) \wedge \sigma_](y) \wedge (x \sim_2 y)) \to x <_1 y$$

Note that models of the conjunction of these formulas will not necessarily have the matching brackets in each equivalence class, as we do not express any stack-like relationship between the linear order and the equivalence relation. However, the projection will result in a string that has an equal number of opening and closing brackets, with every prefix having at least as many opening brackets as there are closing ones, which ensures well-nestedness.

9.4 Normal form for $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$

Here we show a normal form for $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ akin to the Scott Normal Form, which will break down the satisfiability problem to smaller atomic parts called types. These will be used to speak about universal and existential constraints on (pairs of) elements.

Definition 9.11 (1-type, 2-type)**.** *A* 1-type $\nu(x)$ *over* voc *is a maximal consistent conjunction of atomic and negated atomic formulas over* voc *with the free variable* x*. A* 2-type $\theta(x, y)$ *is defined similarly.*

Every element d of a data word $\mathcal{D}$ *realizes* a unique 1-type consisting of the atomic and negated atomic formulas that it satisfies in $\mathcal{D}$. That is, the letter of d appears positively in its 1-type, and all other letters appear with negation. Similarly, every pair (d, d') of elements realizes a unique 2-type which specifies whether $d \leq_1 d'$, etc. Note that when this 2-type is restricted to its x and y components, it determines the 1-types realized by d and d'.

Example 9.12. *The* 1*-type realized by the element* d *of the data* $\mathcal{D}$ *from Example 9.6 in Figure 9.1 is* $\{\xi_2(x), \neg\xi_1(x)\}$*. The* 2*-type realized by the pair* (c, d) *is given by the positive atomic formulas (we omit the negated ones)* $\{x <_1 y, x \prec_2 y, S_2(x, y), \xi_1(x), \xi_2(y)\}$*.*

The following normal form theorem has many technical details, but it essentially states that any formula in $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ can be converted to an equivalent (in a sense) formula which uses types.

Theorem 9.13 (Normal Form)**.** *Let* ψ *be a sentence in* $\mathrm{FO}^2(\mathrm{voc}_{\mathrm{DW}}(\Sigma))$*. Then there exist*

- *numbers* $A, B, C \in \mathbb{N}$*,*
- *an alphabet* $\Xi = \{\xi_a \mid 1 \leq a \leq A\}$*,*
- *a set of 2-types* $\Theta = \Theta_\forall \cup \Theta_\exists$ *with* $\Theta_\exists = \{\theta_{abc} \mid a \in [A], b \in [B], c \in [C]\}$*,*
- *a formula* $\varphi \in \mathrm{FO}^2(\mathrm{voc}_{\mathrm{DW}}(\Xi))$ *of the form* $\varphi = \varphi_\forall \wedge \varphi_\exists$

where

$$\varphi_\forall = \forall x \forall y\, \chi(x, y) \quad \text{with} \quad \chi(x, y) = \bigvee_{\theta \in \Theta_\forall} \theta(x, y),$$

$$\varphi_\exists = \varphi_\varepsilon \wedge \forall x \bigwedge_{a=1}^{A} \xi_a(x) \to \bigwedge_{b=1}^{B} \exists y \bigvee_{c=1}^{C} \theta_{abc}(x, y),$$

$$\varphi_\varepsilon = \begin{cases} \mathit{True}, & \emptyset_{\mathrm{voc}_{\mathrm{DW}}(\Sigma)} \models_{\mathrm{DW}(\Sigma)} \psi \\ \exists x\, (\mathit{True}), & \emptyset_{\mathrm{voc}_{\mathrm{DW}}(\Sigma)} \not\models_{\mathrm{DW}(\Sigma)} \psi \end{cases}$$

and a letter-to-letter substitution $h : \Xi \to \Sigma$ *whose extension to languages is* $\hat{h} : 2^{\Xi^\star} \to 2^{\Sigma^\star}$*, such that* $L(\psi) = \hat{h}(L(\varphi))$*. Moreover, given* ψ*, the formula* φ *is computable in* ExpSpace *and is of length exponential in* $|\psi|$*.*

Proof of Theorem 9.13 We denote the class of all finite structures over a vocabulary voc by Str(voc). To define h, we first need to introduce two functions (called translations), trans_1 and trans_2. For simplicity we assume that the empty word satisfies ψ, and therefore $\varphi_\varepsilon = \mathit{True}$. For the other case, we need to change the following by conjoining each of φ^0, $\varphi^1_\exists$, and $\varphi^2_\exists$ with $\varphi_\varepsilon = \exists x\,(\mathit{True})$.

The translation trans_1**.** Let

$$\varphi^0 = \forall x \forall y\, \chi^0(x,y) \wedge \bigwedge_{b=1}^{B} \forall x \exists y\, \chi^0_b(x,y)$$

be the Scott Normal Form of ψ (Theorem 9.3). The formula φ^0 is over the vocabulary $\text{voc}_{\text{DW}}(\Sigma) \cup \text{voc}_{\text{SNF}}$ where voc_{SNF} is a set of fresh unary relations. The formulas χ^0, and χ^0_b are quantifier-free. The length of φ^0 is linear in that of ψ.

For every model $\mathcal{D} \in \text{DW}(\Sigma)$ of ψ, there is a unique expansion of $\mathcal{D}$ which satisfies φ^0. Let

$$\text{trans}_1 : \text{Str}(\text{voc}_{\text{DW}}(\Sigma) \cup \text{voc}_{\text{SNF}}) \to \text{DW}(\Sigma)$$

be the function which takes a $\text{Str}(\text{voc}_{\text{DW}}(\Sigma) \cup \text{voc}_{\text{SNF}})$-structure to its reduct to $\text{voc}_{\text{DW}}(\Sigma)$. The following hold:

(i^0_1) for every $\mathcal{E} \in \text{Str}(\text{voc}_{\text{DW}}(\Sigma) \cup \text{voc}_{\text{SNF}})$, if $\mathcal{E} \models \varphi^0$ then $\text{trans}_1(\mathcal{E}) \models_{\text{DW}(\Sigma)} \psi$, and

(ii^0_1) for every $\mathcal{D} \in \text{DW}(\Sigma)$, if $\mathcal{D} \models_{\text{DW}(\Sigma)} \psi$ then there exists $\mathcal{E} \in \text{Str}(\text{voc}_{\text{DW}}(\Sigma) \cup \text{voc}_{\text{SNF}})$ such that $\text{trans}_1(\mathcal{E}) = \mathcal{D}$ and $\mathcal{E} \models \varphi^0$.

Making the types explicit Next we define a formula φ^1 which is equivalent to φ^0. Let A be a finite set such that $\{\nu_a \mid a \in A\}$ is the set of 1-types over $\text{voc}_{\text{DW}}(\Sigma) \cup \text{voc}_{\text{SNF}}$. Every quantifier-free formula is equivalent to a disjunction of 2-types. Hence, there is a set C whose size is at most the number of 2-types over $\text{voc}_{\text{DW}}(\Sigma) \cup \text{voc}_{\text{SNF}}$ such that every conjunct $\forall x \exists y\, \chi^0_b(x,y)$ is equivalent to

$$\forall x \bigwedge_{a=1}^{A} \nu_a(x) \to \exists y \bigvee_{c=1}^{C} \beta_{abc}(x,y)$$

where $\beta_{abc}(x,y)$ is 2-type over $\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}}$ for every a, b, and c. There is a set $\Theta^{\beta}_{\forall}(x,y)$ of 2-types over $\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}}$ such that

$$\begin{aligned} \chi^0(x,y) &\equiv \bigvee_{\beta \in \Theta^{\beta}_{\forall}} \beta(x,y), \\ \varphi^0 &\equiv \forall x \forall y \bigvee_{\beta \in \Theta^{\beta}_{\forall}} \beta(x,y) \wedge \bigwedge_{b=1}^{B} \forall x \bigwedge_{a=1}^{A} \nu_a(x) \rightarrow \exists y \bigvee_{c=1}^{C} \beta_{abc}(x,y). \end{aligned}$$

Let $\varphi^1 = \varphi^1_{\forall} \wedge \varphi^1_{\exists}$, where

$$\begin{aligned} \varphi^1_{\forall} &= \forall x \forall y \bigvee_{\beta \in \Theta^{\beta}_{\forall}} \beta(x,y), \\ \varphi^1_{\exists} &= \forall x \bigwedge_{a=1}^{A} \nu_a(x) \rightarrow \bigwedge_{b=1}^{B} \exists y \bigvee_{c=1}^{C} \beta_{abc}(x,y). \end{aligned}$$

We have $\varphi^1 \equiv \varphi^0$ and from (i^0_1) and (ii^0_1):

- (i^1_1) for every $\mathcal{E} \in \mathrm{Str}(\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}})$, if $\mathcal{E} \models \varphi^1$ then $\mathrm{trans}_1(\mathcal{E}) \models \psi$, and
- (ii^1_1) for every $\mathcal{D} \in \mathrm{DW}(\Sigma)$, if $\mathcal{D} \models_{\mathrm{DW}(\Sigma)} \psi$ then there exists $\mathcal{E} \in \mathrm{Str}(\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}})$ such that $\mathrm{trans}_1(\mathcal{E}) = \mathcal{D}$ and $\mathcal{E} \models \varphi^1$.

The translation trans_2**.** Let $\Xi = \{\xi_a \mid a \in [A]\}$. For every 2-type β over $\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}}$, let β^{Ξ} be the 2-type over $\mathrm{voc}_{\mathrm{DW}}(\Xi)$ such that:

- For every $\alpha(x,y)$ which is one of $R(x,y)$ or $\neg R(y,x)$ for $R \in \{\leq_1, \precsim_2, S_2\}$, $\beta(x,y) \models \alpha(x,y)$ if and only if $\beta^{\Xi}(x,y) \models \alpha(x,y)$.
- For every $a \in [A]$ and $z \in \{x,y\}$, $\beta(x,y) \models \nu_a(z)$ if and only if $\beta^{\Xi}(x,y) \models \xi_a(z)$.

Let $\varphi^2 \in \mathrm{FO}^2(\mathrm{voc}_{\mathrm{DW}}(\Xi))$ be the formula obtained from φ^1 by replacing the 1-types $\nu_a(x)$ with $\xi_a(x)$ and the 2-types $\beta(x,y)$ and $\beta_{abc}(x,y)$ with $\beta^{\Xi}(x,y)$ and $\beta^{\Xi}_{abc}(x,y)$ respectively:

$$\varphi^2 = \varphi^2_{\forall} \wedge \varphi^2_{\exists}$$

where

$$\begin{aligned} \varphi^2_{\forall} &= \forall x \forall y \bigvee_{\beta^{\Xi} \in \Theta^{\Xi}_{\forall}} \beta^{\Xi}(x,y), \\ \varphi^2_{\exists} &= \forall x \bigwedge_{a=1}^{A} \xi_a(x) \rightarrow \bigwedge_{b=1}^{B} \exists y \bigvee_{c=1}^{C} \beta^{\Xi}_{abc}(x,y). \end{aligned}$$

and where $\Theta^{\Xi}_{\forall} = \{\beta^{\Xi} \mid \beta \in \Theta^{\beta}_{\forall}\}$.

Finally, we define the translation

$$\mathrm{trans}_2 : \mathrm{DW}(\Xi) \rightarrow \mathrm{Str}(\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}}).$$

Let $\mathcal{D} \in \mathrm{DW}(\Xi)$ with universe D. We define $\mathrm{trans}_2(\mathcal{D})$ as follows:

- The universe and order relations of $\mathrm{trans}_2(\mathcal{D})$ are identical to those of $\mathcal{D}$.
- For every $\xi_a \in \Xi$ and $d \in D$, if $\mathcal{D} \models \xi_a(d)$ then d has 1-type $\nu_a(x)$ in $\mathrm{trans}_2(\mathcal{D})$.

Observe that for a data word $\mathcal{D} \in \mathrm{DW}(\Xi)$ with universe D, we have for all $d, d' \in D$ and every 2-type β^Ξ, $\mathcal{D} \models \beta^\Xi(d, d')$ if and only if $\mathrm{trans}_2(\mathcal{D}) \models \beta(d, d')$. Hence, from (i_1^1) and (i_1^1):

- (i_2) for every $\mathcal{E} \in \mathrm{DW}(\Xi)$, if $\mathcal{E} \models_{\mathrm{DW}(\Xi)} \varphi^2$ then $\mathrm{trans}_2(\mathcal{E}) \models \varphi^1$, and
- (ii_2) for every $\mathcal{D} \in \mathrm{Str}(\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}})$, if $\mathcal{D} \models \varphi^1$ then there exists $\mathcal{E} \in \mathrm{DW}(\Xi)$ such that $\mathrm{trans}_2(\mathcal{E}) = \mathcal{D}$ and $\mathcal{E} \models_{\mathrm{DW}(\Xi)} \varphi^2$.

Let h be a letter-to-letter substitution given as follows: for every $\xi_a \in \Xi$, take $h(\xi_a) = \sigma_a$, where σ_a is the unique letter in Σ such that $\nu_a(x) \models \sigma_a(x)$. Let $\overline{h}$ be the function $\overline{h} : \mathrm{DW}(\Xi) \to \mathrm{DW}(\Sigma)$ such that for every $\mathcal{D} \in \mathrm{DW}(\Xi)$, $\overline{h}(\mathcal{D}) = \mathcal{E}$, where $\mathcal{E}$ has the same universe and order relations as $\mathcal{D}$, and where the interpretation $\sigma^{\mathcal{E}}$ of $\sigma \in \Sigma$ in $\mathcal{E}$ is $\bigcup_{\xi \in \overline{h}^{-1}(\sigma)} \xi^{\mathcal{D}}$. Note that $\overline{h}$ is the composition of trans_2 and trans_1. Using (i_1^1), (i_2), (ii_1^1), and (ii_2), we have:

- (i_h) for every $\mathcal{E} \in \mathrm{DW}(\Xi)$, $\mathcal{E} \models_{\mathrm{DW}(\Xi)} \varphi^2$ implies $\overline{h}(\mathcal{E}) \models_{\mathrm{DW}(\Sigma)} \psi$, and
- (ii_h) for every $\mathcal{D} \in \mathrm{DW}(\Sigma)$, $\mathcal{D} \models_{\mathrm{DW}(\Sigma)} \psi$ implies the existence of $\mathcal{E} \in \mathrm{DW}(\Xi)$ such that $\overline{h}(\mathcal{E}) = \mathcal{D}$ and $\mathcal{E} \models_{\mathrm{DW}(\Xi)} \varphi^2$.

Let $\hat{h}$ be the function $\hat{h} : 2^{\Xi^\star} \to 2^{\Sigma^\star}$ which transforms every word u in the input language by substituting the letters according to h. Now we can prove that $L(\psi) = \hat{h}(L(\varphi^2))$. Observe that for $\mathcal{E} \in \mathrm{DW}(\Xi)$,

$$\hat{h}(\{\mathrm{string}(\mathcal{E})\}) = \{\mathrm{string}(\overline{h}(\mathcal{E}))\}.$$

Let $u \in \hat{h}(L(\varphi^2))$. There is some $\mathcal{E} \in \mathrm{DW}(\Xi)$ such that $\mathcal{E} \models \varphi^2$ and $\{u\} = \hat{h}(\{\mathrm{string}(\mathcal{E})\})$ and hence $u = \mathrm{string}(\overline{h}(\mathcal{E}))$. By (i_h), $\overline{h}(\mathcal{E}) \models \psi$, and hence $u \in \hat{h}(L(\varphi^2))$.

Conversely, let $u \in L(\psi)$. There is some $\mathcal{D} \in \mathrm{DW}(\Sigma)$ such that $\mathcal{D} \models \psi$ and $u = \mathrm{string}(\mathcal{D})$. By (ii_h), there is $\mathcal{E} \in \mathrm{DW}(\Xi)$ such that $\overline{h}(\mathcal{E}) = \mathcal{D}$ and $\mathcal{E} \models_{\mathrm{DW}(\Xi)} \varphi^2$. Hence, $\mathrm{string}(\mathcal{E}) \in L(\varphi^2)$. We have

$$\hat{h}(\{\mathrm{string}(\mathcal{E})\}) = \{\mathrm{string}(\overline{h}(\mathcal{E}))\} = \{\mathrm{string}(\mathcal{D})\} = \{u\},$$

and hence $u \in \hat{h}(L(\varphi^2))$.

Given ψ, the formula φ^0 can be computed in polynomial time in the length of ψ. The size of $\mathrm{voc}_{\mathrm{SNF}}$ is linear in the length of ψ. W.l.o.g. we can assume that every symbol in $\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}}$ occurs in ψ. Then the number of 1-types and 2-types over $\mathrm{voc}_{\mathrm{DW}}(\Sigma) \cup \mathrm{voc}_{\mathrm{SNF}}$ is at most exponential in the length of ψ, and the formulas φ^1 and φ^2 can be computed in exponential space. The theorem follows with the notation slightly simplified by replacing

- β^{Ξ} with θ,
- β^{Ξ}_{abc} with θ_{abc},
- $\Theta^{\Xi}_{\forall}$ with $\Theta_{\forall}$,
- $\varphi^2_{\exists}$ with $\varphi_\varepsilon \wedge \forall x \bigwedge_{a=1}^{A} \xi_a(x) \to \bigwedge_{b=1}^{B} \exists y \bigvee_{c=1}^{C} \theta_{abc}(x,y)$, and
- $\varphi^2_{\forall}$ with $\forall x \forall y \bigvee_{\theta \in \Theta_\forall} \theta(x,y)$.

□

Before we finish the chapter, we present an example of the normal form.

Example 9.14. *Consider the following formula given in normal form*

$$\begin{aligned}\varphi \;=\; & \forall x \forall y\, \chi(x,y) \wedge \\ & \forall x \Big((\xi_1(x) \to \exists y\, (\theta_1(x,y) \vee \theta_3(x,y))) \wedge \\ & \qquad (\xi_2(x) \to \exists y\, (\theta_2(x,y) \vee \theta_4(x,y))) \Big)\end{aligned}$$

where θ_i, $i \in [4]$ are given as the following 2-types (omitted clauses are negated):

$$\begin{aligned}\theta_1 &= x <_1 y \wedge S_2(x,y) \wedge \xi_1(x) \wedge \xi_2(y) \\ \theta_2 &= y <_1 x \wedge S_2(y,x) \wedge \xi_2(x) \wedge \xi_1(y) \\ \theta_3 &= x <_1 y \wedge \neg S_2(x,y) \wedge x \precsim_2 y \wedge \xi_1(x) \wedge \xi_2(y) \\ \theta_4 &= y <_1 x \wedge \neg S_2(y,x) \wedge y \precsim_2 x \wedge \xi_2(x) \wedge \xi_1(y)\end{aligned}$$

and we define $\chi(x,y)$ as the disjunction of 2-types equivalent to $(\xi_2(x) \wedge \xi_2(y)) \to x \sim_2 y$. A data word $\mathcal{D}$ satisfies φ if and only if $\mathcal{D}$ is the empty structure, or all the following hold:

(i) the largest element of $\leq_1$ has letter ξ_2,

(ii) the smallest element of $\leq_1$ has letter ξ_1,

(iii) all elements with letter ξ_2 have the maximal value, and

(iv) all elements with letter ξ_1 have non-maximal values.

Note that $\mathcal{D}$ from Example 9.6 is a model of φ, and that the projection language $\mathscr{L}_{\mathrm{str}}(\varphi)$ is the regular language with regular expression $\xi_1(\xi_1 + \xi_2)^\star \xi_2 + \varepsilon$.

We have shown that any $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ formula ψ can be transformed into a formula φ in a normal form that will allow us to check satisfaction by checking a universal condition that holds for all elements, and existential conditions. Importantly, these conditions are expressed via 1-types and 2-types, which makes it at least conceivable to treat them as letters by an automaton. However, we are still dealing with data words, which are incompatible with PIAs. Transforming data words into strings will require several technical steps, which we will describe in the next chapter. These transformations will inevitably lose information about the data values; they must in order to maintain finiteness of the alphabet our PIA will run on. Nonetheless, this loss will not compromise our ability to check satisfiability of the original formula. As we will see, it will be enough for the PIA to categorize data values as: equal to the data value of current position, smaller than it by 1, or smaller than it by more than 1.

CHAPTER 10

PIA and FO^2 with Two Orders

Here we put the two previous chapters together and show that PIAs capture the logic $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ in the following sense:

Theorem 10.1. *Let ψ be an $\mathrm{FO}^2(\mathrm{voc}_{\mathrm{DW}}(\Sigma))$-sentence. There is a PIA $\mathcal{A}$ with alphabet Ξ and a letter-to-letter substitution $h : \Xi \to \Sigma$ with extension to languages $\hat{h} : 2^{\Xi^\star} \to 2^{\Sigma^\star}$ such that $\mathscr{L}_{\mathrm{str}}(\psi) = \hat{h}(\mathscr{L}(\mathcal{A}))$.*

In simpler terms, every projection language of a definable data language has a PIA which accepts it, after some substitution of the letters. An automata model for the *projection* language provides a closer point of comparison with classical automata models, which operate on strings, while maintaining the ability to reason about values from infinite domains.

The result relates logical formulas that define data words to strings accepted by PIA, a finite-memory automata model. This poses a challenge in navigating the transition from the formula to the data words it defines, and from data words to string projections while maintaining a parsimonious relationship between the satisfying data words and the accepted inputs of the PIA. Beyond the technical difficulty of making these transitions, for the latter part we also need to reconcile the facts that PIAs have finite memory and that our data words have data values from an infinite domain. We will achieve this by careful bookkeeping of which parts of the formula were already satisfied by the input seen so far, and which remain to be satisfied, while taking advantage of the fact that in $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$, data values that are smaller than the largest two are in a way indistinguishable, and can therefore be simply considered 'small'.

To show this result, we first rewrite ψ into a formula φ in the normal form of Theorem 9.13, where the quantifier-free formulas are disjunctions of 2-types θ. Here is where the letter substitution mentioned in Theorem 10.1 comes from. We then construct the PIA $\mathcal{A}^\varphi$ for which we will have $\mathscr{L}_{\mathrm{str}}(\varphi) = \mathscr{L}(\mathcal{A}^\varphi)$. The main idea is that since a string is in

the projection language if and only if it can be extended with a preorder in a way that satisfies the formula, we will have the automaton $\mathcal{A}^\varphi$ run on its input in iterations, and we will associate a data value with each iteration. Conceptually, $\mathcal{A}^\varphi$ expands its input into a data word by assigning the data value corresponding to each iteration to the positions read in that iteration. In order to argue that $\mathcal{A}^\varphi$ accepts the projection language of φ, we introduce some notions.

First, we define $\mathcal{D}$-*task words* $\mathcal{T}$ which are data words with the same universe as $\mathcal{D}$ that assign *tasks* to every element of $\mathcal{D}$. A task represents an existential constraint of the form $\exists y\, \theta(x, y)$ on x, which is *completed* if it is satisfied and *promised* if not. The alphabet letter of an element in $\mathcal{T}$ indicates a set of assigned tasks and remembers which of them have been completed. We prove in Proposition 10.32 that $\mathcal{D} \models \varphi$ if and only if there is a $\mathcal{D}$-task word $\mathcal{T}$ with certain properties, namely, $\mathcal{T}$ is *complete* and *perfect* . $\mathcal{T}$ is complete if $\mathcal{D}$ fulfills all the assigned tasks, and it is perfect if $\mathcal{D}$ fulfills the universal constraints of φ.

In analogy to the iterations of $\mathcal{A}^\varphi$ we define the *trimming* of a data word $\mathcal{D}$, as the substructure of $\mathcal{D}$ omitting the elements with the largest data values. We further define trimmings of $\mathcal{D}$-task words $\mathcal{T}$ as the task words of the trimming of $\mathcal{D}$. $\mathcal{A}^\varphi$ will essentially try to determine whether a sequence of task word trimmings exists which is consistent with its input, and which ultimately results in a perfect and completed task word.

In order to bridge the gap between PIAs having limited memory and task words not being bounded in size, we will use *extremal strings*, that are bounded in length. Whether $\mathcal{T}$ is complete and whether it is perfect will be expressible in terms of the extremal strings of its trimmings.

Extremal strings are derived from task words $\mathcal{T}$ in two steps. First, the data values are abstracted to obtain the string $\mathsf{abst}(\mathcal{T})$, by omitting the preorder and partitioning the elements of $\mathcal{T}$ into three *layers* depending on their data value: the maximal value, the second to maximal value, and all other values. The alphabet of $\mathsf{abst}(\mathcal{T})$ will consist of pairs of sets of assigned tasks and layers. Second, we take the substring of $\mathsf{abst}(\mathcal{T})$ corresponding to extremal (maximal or minimal) positions in each layer w.r.t. the tasks.

Then the goal is reduced to finding a sequence of consecutive extremal strings which end in an extremal string which is perfect and completed. By definition, two extremal strings will be consecutive if they are the extremal strings of a task word and its trimming, which seemingly implies that we still need to keep an unbounded structure in the background. However, we will show in Lemma 10.46 that checking whether two extremal strings are consecutive can be done syntactically without a concrete task word, allowing us to indeed reduce the problem of determining whether a data word satisfies φ to finding an appropriate sequence of extremal strings (Lemma 10.33).

10.1 Task words

In what follows, we let φ be given in normal form as in Theorem 9.13. Recall that $\varphi = \varphi_\forall \wedge \varphi_\exists$ where

$$\varphi_\exists \quad = \quad \varphi_\varepsilon \wedge \forall x \bigwedge_{a=1}^{A} \xi_a(x) \to \bigwedge_{b=1}^{B} \exists y \bigvee_{c=1}^{C} \theta_{abc}(x,y)$$

As we mentioned, for finding an assignment of data values that satisfies $\varphi_\exists$, we use *task words*, which are data words whose elements are additionally assigned *tasks*. A task represents an existential constraint along with the information on whether it was completed or not. As the structure of $\varphi_\exists$ indicates, every element must satisfy B existential constraints, where each $\exists y \bigvee_{c=1}^{C} \theta_{abc}(x,y)$ for $b \in [B]$ may be satisfied due to some 2-type θ_{abc}. The set of tasks assigned to an element will correspond to a set of 2-types with which the satisfaction of $\varphi_\exists$ may be witnessed. Such sets will be called *set-types*. We will now make these notions precise.

Set-types Recall our notation for the existential constraints

$$\Theta_\exists = \{\theta_{abc} \mid a \in [A], b \in [B], c \in [C]\}.$$

Given $a \in [A]$, an *a-set-type* is a choice of 2-types θ_{abc} for satisfying the right-hand side of the implication for ξ_a. That is, a set of 2-types $\omega \subseteq \Theta_\exists$ that contains some θ_{abc} for every $b \in [B]$. Set-types represent the existential constraints an element needs to fulfill, and they will be used to define task words, which specify whether these constraints are satisfied or not.

Denote by Ω_a the set of a-set-types and let $\Omega = \bigcup_{a\in[A]} \Omega_a$. For a set-type $\omega \in \Omega$, let $\omega(x) = \bigwedge_{\theta\in\omega} \exists y\, \theta(x,y)$ be the existential constraints implied by ω. Note that there is a unique letter, denoted $\xi^\omega \in \Xi$, such that $\omega(x) \models_{\mathrm{DW}(\Xi)} \xi^\omega(x)$.

Example 10.2. *For φ from Example 9.14, we have $A = 2$, $B = 1$, and $C = 2$. The set-types of φ are*

$$\{\theta_{111}\}, \quad \{\theta_{112}\}, \quad \{\theta_{211}\}, \quad \{\theta_{212}\},$$

where

$$\theta_{111} = \theta_1, \quad \theta_{112} = \theta_3, \quad \theta_{211} = \theta_2, \quad \theta_{212} = \theta_4.$$

Hence, we have

$$\begin{aligned} \Omega &= \{\{\theta_1\}, \{\theta_2\}, \{\theta_3\}, \{\theta_4\}\}, \\ \xi^{\{\theta_1\}} &= \xi^{\{\theta_3\}} = \xi_1, \\ \xi^{\{\theta_2\}} &= \xi^{\{\theta_4\}} = \xi_2. \end{aligned}$$

Tasks Tasks carry with them the additional information on whether they were fulfilled or not. As data values are added to a string, tasks can go from being promised to being completed, i.e. go from storing the assumption they will be satisfied in some extension, to having this satisfaction established. Denote

$$\text{Tasks}_\text{C} = \{C_\theta \mid \theta \in \Theta_\exists\}, \quad \text{Tasks}_\text{P} = \{P_\theta \mid \theta \in \Theta_\exists\}, \quad \text{Tasks} = \text{Tasks}_\text{C} \cup \text{Tasks}_\text{P}.$$

We refer to C_θ as *completed tasks* and to P_θ as *promised tasks*. Let $ts \subseteq \text{Tasks}$ and $\omega \in \Omega$. We say a task ts *realizes* ω if for every $\theta \in \omega$, ts contains exactly one of C_θ and P_θ and nothing else. That is, ts indicates a set of assigned tasks and remembers which of them have been completed. Obviously, not all $ts \subseteq \text{Tasks}$ realize a set-type, and we say ts is an Ω-*realization* if there exists a set-type $\omega \in \Omega$ such that ts realizes ω. We denote

$$2_\Omega^{\text{Tasks}} = \{ts \subseteq \text{Tasks} \mid ts \text{ is a } \Omega\text{-realization}\}.$$

If $ts \in 2_\Omega^{\text{Tasks}}$, we denote by $\omega(ts)$ the unique set-type that ts realizes. We denote

$$2_\Omega^{\text{Tasks}_\text{C}} = 2_\Omega^{\text{Tasks}} \cap 2^{\text{Tasks}_\text{C}}, \quad 2_\Omega^{\text{Tasks}_\text{P}} = 2_\Omega^{\text{Tasks}} \cap 2^{\text{Tasks}_\text{P}}.$$

Example 10.3. *The set-types Ω from Example 10.2 are singletons, thus each $ts \in 2_\Omega^{\text{Tasks}}$ is a singleton as well. Let $ts_i^C = \{C_{\theta_i}\}$ and $ts_i^P = \{P_{\theta_i}\}$ for $i \in [4]$. Then we have*

$$2_\Omega^{\text{Tasks}} = \{ts_i^C \mid i \in [4]\} \cup \{ts_i^P \mid i \in [4]\},$$

and $\{C_{\theta_i}\}$ and $\{P_{\theta_i}\}$ are $\{\theta_i\}$-realizations for $i \in [4]$.

We first define task words in the context of some data word $\mathcal{D}$. A $\mathcal{D}$-task word essentially assigns tasks to the elements of $\mathcal{D}$ by assigning to each $d \in \mathcal{D}$, instead of a letter ξ_a, a task ts that realizes an a-set-type ω that contains C_θ for each $\theta \in \omega$ that d satisfies, and P_θ for the remaining $\theta \in \omega$.

Definition 10.4 (Task word)**.** *Let $\mathcal{D}$ be a data word over Ξ. A* $\mathcal{D}$-task word *is a data word $\mathcal{T}$ over 2_Ω^{Tasks} which has the same universe and order relations as $\mathcal{D}$, where for every $d \in D$ with $\mathcal{T} \models ts(d)$,*

1. *$\mathcal{D} \models \xi^{\omega(ts)}(d)$, and*
2. *for every $\theta \in \omega(ts)$, $P_\theta \in ts$ if and only if $\mathcal{D} \models \neg\exists y\, \theta(d, y)$.*

A task word *$\mathcal{T}$ is a $\mathcal{D}$-task word for some data word $\mathcal{D}$, and a task word is* completed *if*

$$\mathcal{T} \models \varphi_\varepsilon \wedge \forall x \bigvee_{ts \in 2_\Omega^{\text{Tasks}_\text{C}}} ts(x).$$

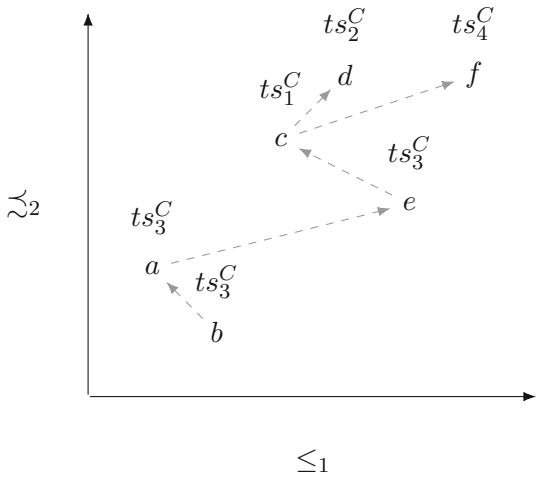


Figure 10.1: The task word $\mathcal{T}$ from Example 10.5.

Example 10.5. *We define a task word $\mathcal{T}$ for the data word $\mathcal{D}$ from Example 9.6, presented in Figure 10.1. The unary relations in the vocabulary of $\mathcal{T}$ are $2^{\text{Tasks}}_{\Omega}$ from Example 10.3. The universe of $\mathcal{T}$ is $\{a, b, c, d, e, f\}$, and the order relations $\leq_1$, $\precsim_2$, and S_2 are the same as in $\mathcal{D}$. The interpretation of the letter ts_1^C is $\{c\}$, that of ts_2^C is $\{d\}$, that of ts_3^C is $\{a, b, e\}$, and that of ts_4^C is $\{f\}$, and the interpretations of the remaining letters are empty. Note that since $\mathcal{D} \models \varphi$, all the existential constraints are satisfied, thus we were able to construct a completed task word $\mathcal{T}$.*

Not surprisingly, the existence of a completed $\mathcal{D}$-task word implies $\mathcal{D}$ satisfies the existential constraints, and vice verse:

Lemma 10.6. *Let $\mathcal{D}$ be a data word over Ξ. There exists a completed $\mathcal{D}$-task word if and only if $\mathcal{D} \models \varphi_{\exists}$.*

Proof. If $\mathcal{D}$ is the empty word $\emptyset_{\text{DW}(\Xi)}$, then the empty word $\emptyset_{2^{\text{Tasks}}_{\Omega}}$ over the vocabulary $2^{\text{Tasks}}_{\Omega}$ is the only $\mathcal{D}$-task word. It is easy to verify that $\emptyset_{2^{\text{Tasks}}_{\Omega}}$ is a completed $\mathcal{D}$-task word if and only if $\mathcal{D} \models \varphi_{\exists}$. From now on, we assume $\mathcal{D}$ is not the empty word. Note that $\mathcal{D} \models \varphi_{\varepsilon}$.

Assume $\mathcal{D} \models \varphi_{\exists}$. We describe a $\mathcal{D}$-task word as follows. Let $\mathcal{T}$ have the same universe and order relations as $\mathcal{D}$. For every $d \in D$, we define the unique $ts_d \in 2^{\text{Tasks}_{\text{C}}}_{\Omega}$ for which $\mathcal{T} \models ts_d(d)$ as follows. Let $a \in [A]$ be such that $\mathcal{D} \models \xi_a(d)$. Since $\mathcal{D} \models \varphi_{\exists}$, we have

$$\mathcal{D} \models \bigwedge_{b=1}^{B} \exists y \bigvee_{c=1}^{C} \theta_{abc}(d, y).$$

Therefore there exist $d_b \in D$ where $b \in B$ and $c_b \in [C]$ where $b \in [B]$ such that $\mathcal{D} \models \theta_{abc_b}(d, d_b)$ for every $b \in [B]$. Let $ts_d = \{C_\theta \mid \theta \in \omega_d\}$, and set $\mathcal{T} \models ts_d(d)$.

The set $\omega_d = \{\theta_{abc_b} \mid b \in [B]\}$ is a set-type and ts_d realizes ω_d. Therefore,

$$\mathcal{T} \models \forall x \bigvee_{ts \in 2^{\text{Tasks}_C}_\Omega} ts(x).$$

For every $d \in D$, $\mathcal{D} \models \xi^\omega(d)$, and for every $\theta \in \omega_d$, we have $P_\theta \notin ts_d$ and $\mathcal{D} \models \exists y\, \theta(d, y)$. Hence, $\mathcal{T}$ is a completed $\mathcal{D}$-task word.

For the other direction, let $\mathcal{T}$ be completed $\mathcal{D}$-task word. Let $d \in D$ and $a \in [A]$ such that $\mathcal{D} \models \xi_a(d)$. Since $\mathcal{T}$ is a completed task word, there exists $ts_d \in 2^{\text{Tasks}_C}_\Omega$ such that $\mathcal{T} \models ts(d)$. Let ω_d be the set-type such that ts realizes ω_d. There exist $c_b \in [C]$ where $b \in [B]$ such that $\omega_d = \{\theta_{abc_b} \mid b \in [B]\}$ and $ts_d = \{C_{\theta_{abc_b}} \mid b \in [B]\}$. Since $\mathcal{T}$ is a task word, $\mathcal{D} \models \exists y\, \theta(d, y)$. Consequently,

$$\mathcal{D} \models \bigwedge_{b=1}^{B} \exists y \bigvee_{c=1}^{C} \theta_{abc}(d, y)$$

for every $d \in D$ and $a \in [A]$, and hence $\mathcal{D} \models \varphi_\exists$.

□

If a completed $\mathcal{D}$-task word exists, i.e. $\mathcal{D} \models \varphi$, we would want the string projection string($\mathcal{D}$) to be accepted by our PIA. For this, we will want to be able to argue that the automaton has a run on string($\mathcal{D}$) from which we could reconstruct $\mathcal{D}$. Toward this, we discuss the sequence of $\mathcal{T}_1, \ldots, \mathcal{T}_n$ leading to a completed task word, where we keep extending the current one with elements that are assigned a new larger data value, while correctly updating promised into completed tasks. The elements of this sequence, which themselves are task words, will be called trimmings as they are thought of as the result of removing the elements with maximal data value from the next task word. So in a sense, this sequence is reversed to how one would obtain it intuitively – which is by starting with a completed task word and iteratively trimming it.

Definition 10.7 (Trimmings)**.** *The* trimming *of a data word $\mathcal{D}$, denoted $\mathcal{D}^{\backslash 1}$, is the substructure of $\mathcal{D}$ induced by removing the elements with the maximal data value. Generally, the* e-trimming *of $\mathcal{D}$ is denoted by $\mathcal{D}^{\backslash e}$, and it is the substructure of $\mathcal{D}$ induced by those elements d for which $value_\mathcal{D}(d) \leq maxval_\mathcal{D} - e$. That is, removing the elements with the e largest data values. Note that $\mathcal{D}^{\backslash 0} = \mathcal{D}$, and if $e \geq maxval_\mathcal{D}$, then $\mathcal{D}^{\backslash e} = \emptyset_\Sigma$.*

Note that this definition simply removes some elements and no more. But for task words, we would like to update the status of each task; that is, if a constraint that is satisfied in the larger structure no longer holds in the trimmed version, we would like to have its task go from complete to promised. For this, we first show the following lemma which states that there is one 'correct' way to update the tasks:

Lemma 10.8. *Let $\mathcal{T}$ be a $\mathcal{D}$-task word for some data word $\mathcal{D}$ over Ξ, and let D_1 denote the universe of $\mathcal{D}^{\backslash 1}$. There is a unique $\mathcal{D}^{\backslash 1}$-task word $\mathcal{T}_1$ such that $\omega(ts) = \omega(ts_1)$ for every $d \in D_1$ and $ts, ts_1 \in 2^{\text{Tasks}}_\Omega$ with $\mathcal{T} \models ts(d)$ and $\mathcal{T}_1 \models ts_1(d)$.*

Proof. First we show the existence of such $\mathcal{T}_1$. We denote the universe of $\mathcal{T}_1$ by T_1. Let $\mathcal{T}_1$ be the $\mathcal{D}^{\setminus 1}$-task word given as follows. For every $d \in T_1$, let $ts \in 2^{\mathrm{Tasks}}_{\Omega}$ be such that $\mathcal{T}_1 \models ts(d)$, and let ts_1 be:

$$ts_1 \quad = \quad \{C_\theta \mid \theta \in \omega(ts), \mathcal{D}^{\setminus 1} \models \exists y\, \theta(d, y)\} \cup \{P_\theta \mid \theta \in \omega(ts), \mathcal{D}^{\setminus 1} \models \neg\exists y\, \theta(d, y)\}.$$

Then $\mathcal{T}_1 \models ts_1(d)$. Since $\omega(ts_1) = \omega(ts)$, Condition 2 in Definition 10.4 holds. Since $\mathcal{T}$ is a $\mathcal{D}$-task word, for every $d \in T_1$ we have $\mathcal{D} \models \xi^{\omega(ts)}(d)$, implying Condition 1 in Definition 10.4 holds.

It remains to show that $\mathcal{T}_1$ is unique. Assume for contradiction that there exists another $\mathcal{D}^{\setminus 1}$-task word $\tilde{\mathcal{T}}$ satisfying the statement of the lemma. Then there is some $d \in D_1$ and distinct $ts_1, \tilde{ts} \in 2^{\mathrm{Tasks}}_{\Omega}$ such that $\mathcal{T}_1 \models ts_1(d)$ and $\tilde{\mathcal{T}} \models \tilde{ts}(d)$. We have $\omega(\tilde{ts}) = \omega(ts) = \omega(ts_1)$, and hence there is $\theta \in \omega(ts)$ such that either $P_\theta \in ts_1 - \tilde{ts}$ or $C_\theta \in ts_1 - \tilde{ts}$. In either case, since $\mathcal{T}_1$ satisfies Condition 2 in Definition 10.4, $\tilde{\mathcal{T}}$ does not satisfy Condition 2, in contradiction to the assumption that $\tilde{\mathcal{T}}$ is a $\mathcal{D}^{\setminus 1}$-task word. □

Now the following is well-defined:

Definition 10.9 (Trimmed task word)**.** *For a $\mathcal{D}$-task word $\mathcal{T}$, denote by $\mathcal{T}^{\setminus\setminus 1}$ the unique $\mathcal{D}^{\setminus 1}$-task word which satisfies Lemma 10.8. By extension, denote by $\mathcal{T}^{\setminus\setminus e}$ the unique $\mathcal{D}^{\setminus e}$-task word obtained from $\mathcal{T}$ by applying trimming e times.*

Note that Definition 10.7 also applies to task words, as they are themselves data words. However, we will never use $\mathcal{T}^{\setminus 1}$ and trimmings of task words will always refer to $\mathcal{T}^{\setminus\setminus 1}$.

The following definition is given in order to make our treatment less cumbersome.

Definition 10.10 (Consecutive task words)**.** *We say that two task words $\mathcal{T}', \mathcal{T}$ are* consecutive *if $\mathcal{T}'$ is a trimming of $\mathcal{T}$.*

Example 10.11. *$\mathcal{D}^{\setminus 1}$ is the substructure of $\mathcal{D}$ from Example 9.6, where the elements with the largest data value, d and f, were removed. This results in the substructure with universe $\{a, b, c, e\}$, presented in Figure 10.2. The $\mathcal{D}^{\setminus 1}$-task word $\mathcal{T}^{\setminus\setminus 1}$ is given as follows. The universe of $\mathcal{T}^{\setminus\setminus 1}$ is $\{a, b, c, e\}$, and the order relations $\leq_1$, $\precsim_2$, and S_2 are the same as in $\mathcal{D}^{\setminus 1}$. Note that the elements d, f, which are missing from $\mathcal{D}^{\setminus 1}$, contributed in $\mathcal{D}$ to the satisfaction of $\varphi_\exists$. Thus, $\mathcal{T}^{\setminus\setminus 1}$ has promised tasks and is no longer a completed task word, with the interpretations: $ts_1^P = \{c\}$, $ts_3^P = \{a, b, e\}$, and the interpretations of the remaining letters being empty. Figure 10.3 presents $\mathcal{T}^{\setminus\setminus 1}$. Note that the tasks for the elements present in both structures realize the same set-types.*

The following proposition simply rephrases Lemma 10.6 to use our notion of consecutiveness.

Proposition 10.12. *For every data word $\mathcal{D}$ over Ξ, $\mathcal{D} \models \varphi_\exists$ if and only if there is a sequence $\mathcal{T}_1 \ldots, \mathcal{T}_n$ of consecutive task words, where $\mathcal{T}_n$ is a completed $\mathcal{D}$-task word.*

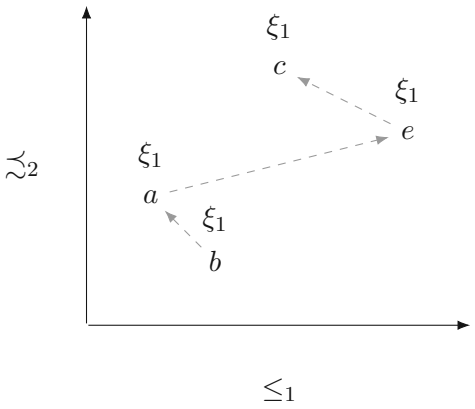


Figure 10.2: The trimming $\mathcal{D}^{\setminus 1}$ from Example 10.11.

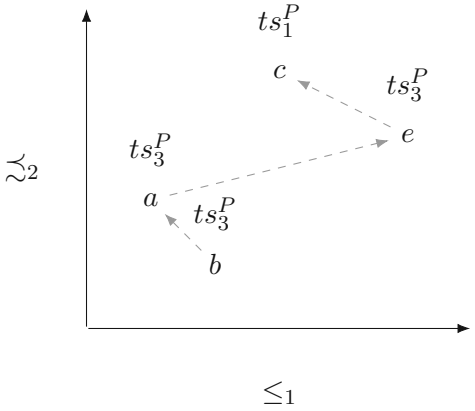


Figure 10.3: The trimmed task word $\mathcal{T}^{\setminus\setminus 1}$ from Example 10.11.

10.2 From task words to extremal strings

To be able to decide the existence of the sequence from Proposition 10.12 with a PIA that can only store a bounded amount of information, we define extremal strings, which will contain all relevant information regarding promised and completed tasks.

Lemma 10.13. *Let $\mathcal{D}$ be a data word and let d, d' be elements with the same 1-type such that $\mathcal{D} \models d \leq_1 d' \wedge d \sim_2 d'$. Let $\theta \in \Theta_\exists$.*

1. *If $\theta(x,y) \models x \leq_1 y$ and $\mathcal{D} \models \exists y\ \theta(d',y)$, then $\mathcal{D} \models \exists y\ \theta(d,y)$.*
2. *If $\theta(x,y) \models y \leq_1 x$ and $\mathcal{D} \models \exists y\ \theta(d,y)$, then $\mathcal{D} \models \exists y\ \theta(d',y)$.*

Proof. 1. Since $\mathcal{D} \models d \leq_1 d' \wedge d \sim_2 d'$, and d, d' have the same 1-type, for any element d'' such that $\mathcal{D} \models d' \leq_1 d''$, the 2-type of (d', d'') is the same as the 2-type of (d, d'').

Since $\theta(x,y) \models x \leq_1 y$, if $\mathcal{D} \models \exists y\ \theta(d',y)$ there exists some $d'' \geq_1 d'$ such that $\mathcal{D} \models \theta(d',d'')$. Therefore also $\mathcal{D} \models \theta(d,d'')$ and $\mathcal{D} \models \exists y\ \theta(d,y)$.

2. Analogous to the previous case.

□

First we define *data abstractions* of task words, that abstract the data values of elements into three layers: the *top layer* for elements with maximal data value, the *second to top layer* for elements with the second largest data value, and the layer consisting of the remaining elements. The notion of data abstraction is inspired by the related notion of interval abstraction [99] in program analysis.

Formally, data abstractions are defined as strings over the alphabet

$$\Gamma = \text{Layers} \times 2^{\text{Tasks}}_{\Omega} \text{ where Layers} = \{1\text{top}, 2\text{top}, \text{rest}\}.$$

The restrictions of Γ to completed and promised tasks are

$$\Gamma_{\text{C}} = \text{Layers} \times 2^{\text{Tasks}_{\text{C}}}_{\Omega}, \qquad \Gamma_{\text{P}} = \text{Layers} \times 2^{\text{Tasks}_{\text{P}}}_{\Omega},$$

while $\Gamma_h = \{h\} \times 2^{\text{Tasks}}_{\Omega}$ is its restriction to some $h \in \text{Layers}$. For a symbol $\gamma = (h, ts)$ in Γ, we denote $ts(\gamma) = ts$ for its task component and $\omega(\gamma) = \omega(ts)$ for the set-type realized by its task component.

Definition 10.14 (Data abstraction)**.** *Let $\mathcal{D}$ be a data words over Ξ and let $\mathcal{T}$ be a $\mathcal{D}$-task word. Let $\mathcal{A}$ be the data word over Γ defined as follows. $\mathcal{A}$ has the same universe and order relations as $\mathcal{T}$. For every $d \in D$, let ts_d be s.t. $\mathcal{T} \models ts_d(d)$, and define $\mathcal{A} \models \gamma_h(d)$, where $\gamma_h = (h, ts_d)$ if and only if*

1. *$h = 1\text{top}$ and $value_{\mathcal{D}}(d) = maxval_{\mathcal{D}}$,*
2. *$h = 2\text{top}$ and $value_{\mathcal{D}}(d) = maxval_{\mathcal{D}} - 1$, or*
3. *$h = \text{rest}$ and $value_{\mathcal{D}}(d) \in [maxval_{\mathcal{D}} - 2]$.*

The data abstraction $\mathsf{abst}(\mathcal{T})$ *is the string projection* $\text{string}(\mathcal{A})$.

As in Definition 9.7, also here we have an embedding induced by the data abstraction, which we denote $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}$.

Example 10.15. *The structure $\mathcal{A}$ from Definition 10.14 for $\mathcal{T}$ from Example 10.5 is presented in Figure 10.4, and we have*

$$\mathsf{abst}(\mathcal{T}) = (\text{rest}, ts_3^C)(\text{rest}, ts_3^C)(2\text{top}, ts_1^C)(1\text{top}, ts_2^C)(\text{rest}, ts_3^C)(1\text{top}, ts_4^C)$$

and for $\mathcal{T}^{\setminus\!\setminus 1}$ from Example 10.11 we have

$$\mathsf{abst}(\mathcal{T}^{\setminus\!\setminus 1}) = (\text{rest}, ts_3^P)(\text{rest}, ts_3^P)(1\text{top}, ts_1^P)(2\text{top}, ts_3^P)$$

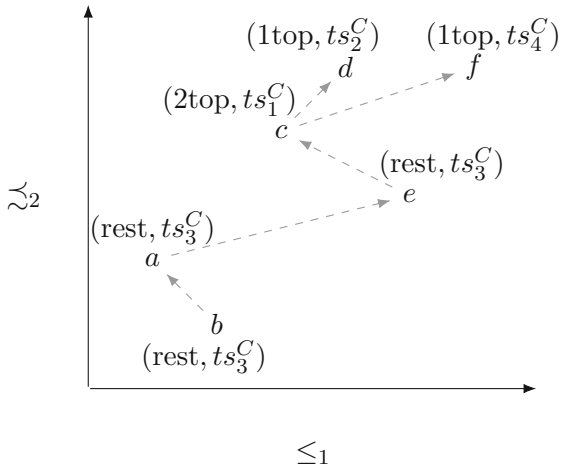


Figure 10.4: The structure $\mathcal{A}$ from Example 10.15.

The embeddings $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}$ *and* $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}\setminus\!\setminus 1}$ *are given by:*

$$\begin{array}{lcllcl}
\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(1) & = & a & \mathsf{Emb}_{\mathsf{abst},\mathcal{T}\setminus\!\setminus 1}(1) & = & a \\
\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(2) & = & b & \mathsf{Emb}_{\mathsf{abst},\mathcal{T}\setminus\!\setminus 1}(2) & = & b \\
\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(3) & = & c & \mathsf{Emb}_{\mathsf{abst},\mathcal{T}\setminus\!\setminus 1}(3) & = & c \\
\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(4) & = & d & \mathsf{Emb}_{\mathsf{abst},\mathcal{T}\setminus\!\setminus 1}(4) & = & e \\
\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(5) & = & e & & & \\
\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(6) & = & f & & &
\end{array}$$

Extremal strings are derived from task words $\mathcal{T}$ by taking their data abstractions, and then taking the substring of $\mathsf{abst}(\mathcal{T})$ corresponding to extremal (maximal or minimal) positions in each layer with respect to the tasks.

Definition 10.16 (ext, extremal strings)**.** *Let* $w \in \Gamma^\star$*. We define subsets of* $[|w|]$*, for every* $h \in \text{Layers}$ *and* $\theta \in \Theta_\exists$*.*

The positions with a specific constraint at a specific layer:

$$\mathsf{Pos}_{h,\theta}(w) = \{\ell \in [|w|] \mid w(\ell) = (h, ts), \theta \in \omega(ts)\}$$

The positions in the rest *layer with a specific promised constraint:*

$$\mathsf{Pos}_{\text{rest},P_\theta}(w) = \{\ell \in [|w|] \mid w(\ell) = (\text{rest}, ts), P_\theta \in ts\}$$

Which are used to define the relevant positions of promised tasks; e.g. if θ *requires there be a larger witness, we would want to keep the largest position since if the task is completed*

for that position, then it also completed for the other positions.

$$\mathsf{ExtPos}_\theta(w) = \begin{cases} \{\ell \mid \ell = \max(\mathsf{Pos}_{\mathrm{rest},P_\theta}(w))\}, & \theta \models x \leq_1 y \\ \{\ell \mid \ell = \min(\mathsf{Pos}_{\mathrm{rest},P_\theta}(w))\}, & \theta \models y <_1 x \end{cases}$$

We also take the extremal positions for a specific layer and constraint, regardless of its status:

$$\mathsf{ExtPos}_{h,\theta}(w) = \{\ell \mid \ell = \max(\mathsf{Pos}_{h,\theta}(w)) \text{ or } \ell = \min(\mathsf{Pos}_{h,\theta}(w))\}$$

And finally, we put it all together:

$$\mathsf{ExtPos}(w) = \bigcup_{\theta\in\Theta_\exists} \left(\bigcup_{h\in\mathrm{Layers}} \mathsf{ExtPos}_{h,\theta}(w) \right) \cup \mathsf{ExtPos}_\theta(w)$$

For $w \in \Gamma^\star$ with $\mathsf{ExtPos}(w) = \{\ell_1, \ldots, \ell_r\}$ and $\ell_1 < \cdots < \ell_r$, We define the function $\mathsf{ext} : \Gamma^\star \to \Gamma^\star$ by

$$\mathsf{ext}(w) = w(\ell_1)\cdots w(\ell_r)$$

i.e. $\mathsf{ext}(w)$ is the substring of w composed of the positions in $\mathsf{ExtPos}(w)$. Again, given w, this operation induces an embedding from the extremal string to w, which we denote $\mathsf{Emb}_{\mathsf{ext},w}$. Note that we have $\mathsf{ext}(\varepsilon) = \varepsilon$, and that for all $\ell \in [|s|]$, we have $\mathsf{Emb}_{\mathsf{ext},w}(\ell) \in \mathsf{ExtPos}(w)$.

At this point we also define the following sets, which will be useful later

$$\mathsf{Pos}_h(w) = \{\ell \in [|w|] \mid w(\ell) \in \Gamma_h\}, \qquad \mathsf{Pos}_{<1\mathrm{top}}(w) = \{\ell \in [|w|] \mid w(\ell) \in \Gamma_{2\mathrm{top}} \cup \Gamma_{\mathrm{rest}}\}$$

Note that for all $w \in \Gamma^\star$, $s = \mathsf{ext}(w)$ implies $\mathsf{ext}(s) = s$.

Let $\mathsf{EXT}(\Gamma) = \{\mathsf{ext}(w) \mid w \in \Gamma^\star\}$. If $s \in \mathsf{EXT}(\Gamma)$, we say s is an extremal string.

Example 10.17. *Let w denote the data abstraction of $\mathcal{T}$ from Example 10.15, and recall it is given by:*

$$w = (\mathrm{rest}, ts_3^C)(\mathrm{rest}, ts_3^C)(2\mathrm{top}, ts_1^C)(1\mathrm{top}, ts_2^C)(\mathrm{rest}, ts_3^C)(1\mathrm{top}, ts_4^C)$$

We have $\mathsf{ExtPos}(w) = \{1, 3, 4, 5, 6\}$, since the letter at position 2 appears both to the left, at position 1, and to the right, at position 5. Let $s = \mathsf{ext}(w)$, then s is the 5-letter string over Γ given by:

$$s = (\mathrm{rest}, ts_3^C)(2\mathrm{top}, ts_1^C)(1\mathrm{top}, ts_2^C)(\mathrm{rest}, ts_3^C)(1\mathrm{top}, ts_4^C)$$

and is the substring obtained from w by removing the non-extremal position 2. Let w' denote the data abstraction of $\mathcal{T}^{\setminus\!\setminus 1}$, recall that:

$$w' = (\mathrm{rest}, ts_3^P)(\mathrm{rest}, ts_3^P)(1\mathrm{top}, ts_1^P)(2\mathrm{top}, ts_3^P).$$

We have $\mathsf{ExtPos}(w') = [4]$. Let $s' = \mathsf{ext}(w')$, then $s' = w'$.

The embedding $\mathsf{Emb}_{\mathsf{ext},w}$ *is given by:*

$$\mathsf{Emb}_{\mathsf{ext},w}(1) = 1 \qquad \mathsf{Emb}_{\mathsf{ext},w}(3) = 4 \qquad \mathsf{Emb}_{\mathsf{ext},w}(5) = 6$$
$$\mathsf{Emb}_{\mathsf{ext},w}(2) = 3 \qquad \mathsf{Emb}_{\mathsf{ext},w}(4) = 5$$

The embedding $\mathsf{Emb}_{\mathsf{ext},w'}$ *is the identity function* $\mathsf{Emb}_{\mathsf{ext},w'}(i) = i$.

In the sequel, we will abuse notation and write $\mathsf{ext}(\mathcal{T})$ to mean $\mathsf{ext}(\mathsf{abst}(\mathcal{T}))$.

We will also be interested in the positions of a string which are not extremal, as the automaton will have to consume these in a way that is consistent with the sequence of extremal strings it is trying to find.

Definition 10.18. *Let* s *be an extremal string and let* $\ell \in [|s|]$. *Define*

$$\Gamma^{\mathsf{not\,ext}}(s,\ell) = \{\gamma \in \Gamma \mid \mathsf{ext}(s) = \mathsf{ext}(s(1)\cdots s(\ell-1)\gamma s(\ell)\cdots s(|s|))\}$$

i.e. the set of letters that can augment s *at position* ℓ *without being extremal. For* $h \in$ Layers, *define*

$$\Gamma_h^{\mathsf{not\,ext}}(s,\ell) = \Gamma_h \cap \Gamma^{\mathsf{not\,ext}}(s,\ell)$$

Since we have a bounded number of constraint and a bounded number of times a constraint can appear in extremal strings, we have the following crucial property:

Lemma 10.19. $\mathsf{EXT}(\Gamma) \subseteq \Gamma^{7\cdot|\Theta_\exists|}$.

Proof. Let $s = \mathsf{ext}(w)$ be an extremal string. For every $\theta \in \Theta_\exists$ and $h \in$ Layers, we have $|\mathsf{ExtPos}_{h,\theta}(w)| \leq 2$ and $|\mathsf{ExtPos}_\theta(w)| \leq 1$. Hence $|\mathsf{ExtPos}(w)| \leq 7 \cdot |\Theta_\exists|$. □

Analogously as for task words, an extremal string is complete if all its tasks are completed.

Definition 10.20 (Completed extremal strings)**.** *For an extremal string* s, *if* $s \in \Gamma_C^+$, *i.e. if all the tasks appearing in* s *are completed, or if* $s = \varepsilon$ *and* $\emptyset_{\mathrm{DW}(\Sigma)} \models_{\mathrm{DW}(\Sigma)} \psi$, *we say* s *is a* completed extremal string.

Consecutive extremal strings are the extremal strings of a task word and its trimming. Note that the same pair of consecutive extremal strings may be obtained from different task words and their trimmings, as they may contain different non extremal positions.

Definition 10.21 (Consecutive extremal strings)**.** *A pair* (s', s) *of extremal strings is* consecutive *if there is a task word* $\mathcal{T}$ *s.t.* $s' = \mathsf{ext}(\mathcal{T}^{\setminus\!\setminus 1})$ *and* $s = \mathsf{ext}(\mathcal{T})$.

The crucial property of consecutive extremal strings will be that they define a partial embedding between their positions independently from the choice of the task word, but it is technically challenging to show this, so we reserve the proof to a later section.

Example 10.22. *Note that the pair* (s', s) *from Example 10.17 is consecutive, since* $s' = \mathsf{ext}(\mathcal{T}^{\setminus\!\setminus 1})$ *and* $s = \mathsf{ext}(\mathcal{T})$.

Now we can rephrase Proposition 10.12 to use the terminology of extremal strings:

Proposition 10.23. *There is* $\mathcal{D} \models \varphi_\exists$ *if and only if there is a sequence of consecutive extremal strings where the last one is completed.*

10.2.1 Perfect extremal strings for $\varphi_\forall$

We now focus on the subformula $\varphi_\forall$. To ensure its satisfaction, we introduce *perfect* extremal strings. Let $\alpha, \beta \in \Gamma$ where either α, β, or both, is in $\Gamma_{1\text{top}}$. The formula $\text{perfect}_{\alpha,\beta}(x, y)$ has a very syntactic definition below, which extracts the 2-type of elements in a data word according to the letters α and β assigned to them under the data abstraction.

Definition 10.24 (The formula $\text{perfect}_{\alpha,\beta}(x, y)$)**.** *Let* $\alpha, \beta \in \Gamma$ *with either* α *or* β*, or both, in* $\Gamma_{1\text{top}}$*. Let* $\alpha = (h_\alpha, ts_\alpha)$ *and* $\beta = (h_\beta, ts_\beta)$*. We define*

1. $\text{perfect}_\alpha(x) = \xi^{\omega(ts_\alpha)}(x)$;
2. $\text{perfect}_\beta(y) = \xi^{\omega(ts_\beta)}(y)$;
3. $\text{perfect}_{\alpha,\beta,\leq_1}(x, y) = x <_1 y$;
4. $$\text{perfect}_{\alpha,\beta,\precsim_2}(x, y) = \begin{cases} x \precsim_2 y, & h_\alpha \neq 1\text{top} \\ y \precsim_2 x, & h_\beta \neq 1\text{top} \\ x \sim_2 y, & \textit{Otherwise}; \end{cases}$$
5. $$\text{perfect}_{\alpha,\beta,S_2}(x, y) = \begin{cases} S_2(x, y), & h_\alpha = 2\text{top} \\ S_2(y, x), & h_\beta = 2\text{top} \\ \neg S_2(x, y), & h_\alpha = \text{rest} \\ \neg S_2(y, x), & h_\beta = \text{rest} \\ \textit{True}, & \textit{Otherwise} \end{cases}$$

Define $\text{perfect}_{\alpha,\beta}(x, y)$ *in the vocabulary* $\text{FO}^2(\text{voc}_{\text{DW}}(\Xi))$ *as the formula:*

$$\text{perfect}_\alpha(x) \wedge \text{perfect}_\beta(y) \wedge \bigwedge_{bin \in \{\leq_1, \precsim_2, S_2\}} \text{perfect}_{\alpha,\beta,bin}(x, y).$$

It follows from the above definition that over data words, $\text{perfect}_{\alpha,\beta}(x, y)$ implies for every atomic formula either itself or its negation.

Observation 10.25. *Let $\alpha, \beta \in \Gamma$ such that either α, β, or both are in $\Gamma_{1\text{top}}$. There exists a 2-type $\theta(x,y)$ such that* $\text{perfect}_{\alpha,\beta}(x,y) \equiv_{\text{DW}(\Xi)} \theta(x,y)$.

As a result, the 2-type of elements in a task word (where at least one of them has maximal data value) can be described by the appropriate $\text{perfect}_{\alpha,\beta}$ formula. We first show that a data word satisfies each formula $\text{perfect}_{w(\ell_1),w(\ell_2)}(d_1,d_2)$ induced by a such a pair of elements. Since it is essentially equivalent to their 2-type, this is not surprising:

Lemma 10.26. *Let $\mathcal{T}$ be a $\mathcal{D}$-task word with universe D and let $w = \mathsf{abst}(\mathcal{T})$. Let $d_1, d_2 \in D$ be such that $\mathcal{D} \models d_1 <_1 d_2$ and at least one of them has the largest data value, i.e. $maxval_{\mathcal{D}} \in \{value_{\mathcal{D}}(d_1), value_{\mathcal{D}}(d_2)\}$. Let ℓ_1 and ℓ_2 be such that d_1 is mapped to position ℓ_1 in $\mathsf{abst}(\mathcal{T})$ and d_2 is mapped to position ℓ_2 in $\mathsf{abst}(\mathcal{T})$. Then $\ell_1 < \ell_2$ and $\mathcal{D} \models \text{perfect}_{w(\ell_1),w(\ell_2)}(d_1,d_2)$.*

Proof. It is given that $\mathcal{D} \models \text{perfect}_{w(\ell_1),w(\ell_2),\leq_1}(d_1,d_2)$. Since $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}$ is order-preserving and $\mathcal{D} \models d_1 <_1 d_2$, we get $\ell_1 < \ell_2$. Since $\mathcal{T}$ is a task word, there exist $ts_1, ts_2 \in 2^{\text{Tasks}}_{\Omega}$ such that $\mathcal{T} \models ts_1(d_1)$ and $\mathcal{T} \models ts_2(d_2)$. By definition of a task word, we have that $\mathcal{D} \models \xi^{\omega(ts_1)}(d_1)$ and $\mathcal{D} \models \xi^{\omega(ts_2)}(d_2)$, and therefore $\mathcal{D} \models \text{perfect}_{w(\ell_1)}(d_1) \wedge \text{perfect}_{w(\ell_2)}(d_2)$.

We consider one of the cases for $\text{perfect}_{w(\ell_1),w(\ell_2),\precsim_2}(x,y)$. The other cases can be treated analogously. If $\text{perfect}_{w(\ell_1),w(\ell_2),\precsim_2}(x,y) = x \precsim_2 y$ then $w(\ell_1) \notin \Gamma_{1\text{top}}$ while $w(\ell_2) \in \Gamma_{1\text{top}}$. By definition of $w = \mathsf{abst}(\mathcal{T})$, we have that $value_{\mathcal{D}}(\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\ell_1)) < maxval_{\mathcal{D}}$ and $value_{\mathcal{D}}(\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\ell_2)) = maxval_{\mathcal{D}}$. Hence, $\mathcal{D} \models \text{perfect}_{w(\ell_1),w(\ell_2),\precsim_2}(d_1,d_2)$.

We consider one of the cases for $\text{perfect}_{w(\ell_1),w(\ell_2),S_2}(x,y)$. The other cases can be treated analogously. If $\text{perfect}_{w(\ell_1),w(\ell_2),S_2}(x,y) = \neg S_2(y,x)$ then $w(\ell_1) \in \Gamma_{1\text{top}}$ while $w(\ell_2) \in \Gamma_{\text{rest}}$. By definition of $w = \mathsf{abst}(\mathcal{T})$, we have that $value_{\mathcal{D}}(\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\ell_1)) = maxval_{\mathcal{D}}$ and $value_{\mathcal{D}}(\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\ell_2)) \leq maxval_{\mathcal{D}} - 2$. Therefore, we have $\mathcal{D} \models \text{perfect}_{w(\ell_1),w(\ell_2),S_2}(d_1,d_2)$. □

This leads us to the definition of perfect strings and task words:

Definition 10.27 (Perfect string, perfect task word)**.** *Let $w \in \Gamma^\star$. We say w is a* perfect string *if for every two positions $\ell_1 < \ell_2$ in w such that $\{\ell_1, \ell_2\} \cap \mathsf{Pos}_{1\text{top}}(w) \neq \emptyset$, it holds that*

$$\text{perfect}_{w(\ell_1),w(\ell_2)}(x,y) \models_{\text{DW}(\Xi)} \chi(x,y) \wedge \chi(y,x).$$

Note that the empty string ε is perfect. A task word $\mathcal{T}$ is perfect *if it is either empty, or if $\mathsf{ext}(\mathcal{T})$ is perfect and its trimming is perfect.*

Note the recursive nature of the definition.

Example 10.28. *Let $\alpha = (2\text{top}, ts_1^C)$ and $\beta = (1\text{top}, ts_2^C)$. Then $\text{perfect}_{\alpha,\beta}(x,y)$ is given by:*

$$\text{perfect}_{\alpha,\beta}(x,y) = \xi_1(x) \wedge \xi_2(y) \wedge (x <_1 y) \wedge (x \precsim_2 y) \wedge S_2(x,y).$$

The 2-type θ to which $\mathrm{perfect}_{\alpha,\beta}(x,y)$ *is equivalent over* $\mathrm{DW}(\Xi)$ *is given by the*

$$\mathrm{perfect}_{\alpha,\beta}(x,y) \wedge \neg\xi_2(x) \wedge \neg\xi_1(y) \wedge (y \not<_1 x) \wedge (y \not\lesssim_2 x) \wedge \neg S_2(y,x).$$

We have that w from Example 10.17 is a perfect string, and

$$\mathrm{perfect}_{\alpha,\beta}(x,y) \models_{\mathrm{DW}(\Xi)} \chi(x,y) \wedge \chi(y,x).$$

Next we show that $\mathrm{perfect}_{\alpha,\beta}(x,y)$ implies that the universal constraints are satisfied:

Lemma 10.29. *Let $\alpha, \beta \in \Gamma$ such that at least one of them is in $\Gamma_{1\mathrm{top}}$ and such that*

$$\mathrm{perfect}_{\alpha,\beta}(x,y) \not\models_{\mathrm{DW}(\Xi)} \chi(x,y) \wedge \chi(y,x).$$

Then it holds that

$$\exists x\, \exists y\, \mathrm{perfect}_{\alpha,\beta}(x,y) \models_{\mathrm{DW}(\Xi)} \neg\varphi_\forall.$$

Proof. Since $\mathrm{perfect}_{\alpha,\beta}(x,y) \not\models_{\mathrm{DW}(\Xi)} \chi(x,y)\wedge\chi(y,x)$, there exists $\mathcal{D}$ and elements $d_1, d_2 \in D$ such that $\mathcal{D} \models \mathrm{perfect}_{\alpha,\beta}(d_1,d_2)$ and $\mathcal{D} \not\models \chi(d_1,d_2) \wedge \chi(d_2,d_1)$. For every data word $\mathcal{D}'$ and every two elements $d_1', d_2' \in D'$ such that $\mathcal{D}' \models \mathrm{perfect}_{\alpha,\beta}(d_1',d_2')$, the 2-type of (d_1',d_2') is the same as the 2-type of (d_1,d_2) by Observation 10.25. Denote this 2-type by $\theta(x,y)$. Since $\mathcal{D} \not\models \chi(d_1,d_2) \wedge \chi(d_2,d_1)$, we have that either $\theta(x,y) \notin \Theta_\forall$ or $\theta(y,x) \notin \Theta_\forall$ and therefore also $\mathcal{D}' \not\models \chi(d_1',d_2') \wedge \chi(d_2',d_1')$. □

To guarantee that χ is satisfied, we will use formulas of the form $\mathrm{perfect}_{\alpha,\beta}(x,y)$ arising from all the (abstractions and extremal strings of the) trimmings in the sequence considered. This will cover all pairs of elements d_1, d_2 of the data word, since for every pair there is a (possibly iterated) trimming in which both d_1 and d_2 appear, and one of them has the maximal data value. First we show the following:

Lemma 10.30. *Let $w \in \Gamma^+$. Then w is perfect if and only if $\mathsf{ext}(w)$ is perfect.*

Proof. Let $s = \mathsf{ext}(w)$.

Assume that w is perfect. Let $\ell_1 < \ell_2$ be positions in s such that at least one of $s(\ell_1)$, $s(\ell_2)$ is in $\Gamma_{1\mathrm{top}}$. For $i = 1, 2$, let $\ell_i' = \mathsf{Emb}_{\mathsf{ext},w}(\ell_i)$. We have $w(\ell_i') = s(\ell_i)$ and $\ell_1' < \ell_2'$. Therefore

$$\mathrm{perfect}_{s(\ell_1),s(\ell_2)}(x,y) = \mathrm{perfect}_{w(\ell_1'),w(\ell_2')}(x,y).$$

Since w is perfect, by definition we have that

$$\mathrm{perfect}_{s(\ell_1),s(\ell_2)}(x,y) \models_{\mathrm{DW}(\Xi)} \chi(x,y) \wedge \chi(y,x).$$

Now assume that s is perfect. Let $\ell_1' < \ell_2'$ be positions in w such that at least one of $w(\ell_1')$ and $w(\ell_2')$ is in $\Gamma_{1\mathrm{top}}$. Denote $\ell_1'' = \min\{\ell \mid w(\ell) = w(\ell_1')\}$ and $\ell_2'' = \max\{\ell \mid w(\ell) = w(\ell_2')\}$

and note that $\ell''_1, \ell''_2 \in \mathsf{ExtPos}(w)$. For $i = 1, 2$, let ℓ_i be such that $\ell''_i = \mathsf{Emb}_{\mathsf{ext},w}(\ell_i)$. We have $w(\ell'_i) = w(\ell''_i) = s(\ell_i)$ for $i = 1, 2$ and $\ell_1 < \ell_2$. Hence,

$$\mathrm{perfect}_{w(\ell'_1),w(\ell'_2)}(x, y) = \mathrm{perfect}_{s(\ell_1),s(\ell_2)}(x, y),$$

and since s is perfect,

$$\mathrm{perfect}_{w(\ell'_1),w(\ell'_2)}(x, y) \models_{\mathrm{DW}(\Xi)} \chi(x, y) \wedge \chi(y, x).$$

□

Finally, we can characterize the satisfaction of the universal constraints in terms of task words.

Lemma 10.31. *Let $\mathcal{T}$ be a $\mathcal{D}$-task word. $\mathcal{T}$ is perfect if and only if $\mathcal{D} \models \varphi_\forall$.*

Proof. Assume $\mathcal{T}$ is perfect. Let d_1 and d_2 be distinct elements of $\mathcal{D}$. We show $\mathcal{D} \models \chi(d_1, d_2)$. Let

$$e = \mathit{maxval}_{\mathcal{D}} - \max\{\mathit{value}_{\mathcal{D}}(d_1), \mathit{value}_{\mathcal{D}}(d_2)\}.$$

Denote the universe of $\mathcal{D}^{\setminus e}$ by D'. Note that d_1 and d_2 are in D' and that one of them has maximal data value for that substructure, i.e.

$$\mathit{maxval}_{\mathcal{D}^{\setminus e}} \in \{\mathit{value}_{\mathcal{D}^{\setminus e}}(d_1), \mathit{value}_{\mathcal{D}^{\setminus e}}(d_2)\}.$$

Since $\mathcal{T}$ is a perfect task word, we have that $\mathsf{ext}(\mathcal{T}^{\setminus\!\setminus e})$ is a perfect string, and by Lemma 10.30, so is the string before taking the extremal substring $w_e = \mathsf{abst}(\mathcal{T}^{\setminus\!\setminus e})$. By Lemma 10.26, either $\mathcal{D}^{\setminus e} \models \mathrm{perfect}_{w_e(\ell_1),w_e(\ell_2)}(d_1, d_2)$ or $\mathcal{D}^{\setminus e} \models \mathrm{perfect}_{w_e(\ell_2),w_e(\ell_1)}(d_2, d_1)$. In either case, since $\mathsf{abst}(\mathcal{T}^{\setminus\!\setminus e})$ is perfect, we have that $\mathcal{D}^{\setminus e} \models \chi(d_1, d_2)$. Since $\chi(x, y)$ is quantifier-free and $\mathcal{D}^{\setminus e}$ is a substructure of $\mathcal{D}$, we also have that $\mathcal{D} \models \chi(d_1, d_2)$.

For the other direction, assume $\mathcal{D} \models \varphi_\forall$. Assume for contradiction that there is an e such that $\mathsf{ext}(\mathcal{T}^{\setminus\!\setminus e})$ is not perfect. Then by Lemma 10.30, $w_e = \mathsf{abst}(\mathcal{T}^{\setminus\!\setminus e})$ is also not perfect. That is, there exist positions $\ell_1 < \ell_2$ in w_e such that at least one of $w_e(\ell_1), w_e(\ell_2)$ is in $\Gamma_{1\mathrm{top}}$, and such that

$$\mathrm{perfect}_{w_e(\ell_1),w_e(\ell_2)}(x, y) \not\models_{\mathrm{DW}(\Xi)} \chi(x, y) \wedge \chi(y, x).$$

For $i = 1, 2$, let $d_i = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus\!\setminus e}}(\ell_i)$ be the elements corresponding to these positions. We have $d_1 < d_2$. By the definition of data abstraction, since one of the positions carries a letter from $\Gamma_{1\mathrm{top}}$, we have that one of these elements has maximal data value, i.e. $\mathit{maxval}_{\mathcal{D}^{\setminus e}} \in \{\mathit{value}_{\mathcal{D}^{\setminus e}}(d_1), \mathit{value}_{\mathcal{D}^{\setminus e}}(d_2)\}$. By Lemma 10.26, we have

$$\mathcal{D}^{\setminus e} \models \mathrm{perfect}_{w_e(\ell_1),w_e(\ell_2)}(d_1, d_2)$$

and by applying Lemma 10.29 with $\alpha = w_e(\ell_1)$ and $\beta = w_e(\ell_2)$, we have that $\mathcal{D}^{\setminus e} \not\models \varphi_\forall$. Since $\mathcal{D}^{\setminus e}$ is a substructure of $\mathcal{D}$ and $\varphi_\forall$ is universal, we also have that $\mathcal{D} \not\models \varphi_\forall$ in contradiction to our assumption. □

Putting together the universal and existential constraints, as a corollary of Lemma 10.31 and Lemma 10.6, we get the following characterization of satisfiability:

Proposition 10.32 (Characterization of satisfiability in terms of task words)**.** *For every data word $\mathcal{D} \in \mathrm{DW}(\Xi)$, $\mathcal{D} \models \varphi$ if and only if there exists a perfect completed $\mathcal{D}$-task word.*

And putting the above with Proposition 10.23 gives us the goal our automaton will try to achieve:

Proposition 10.33. *There is $\mathcal{D} \models \varphi$ if and only if there is a sequence of consecutive perfect extremal strings where the last one is completed.*

We still have a major hurdle. If we would like to have a PIA which stores extremal strings, we need to find a way to check their consecutiveness without using a concrete task word. In the next section, we lay the groundwork for a merely syntactic characterization of consecutive extremal strings. This is probably the most technically challenging step, and it involves closely inspecting embeddings induced by the abst and ext operations.

10.3 Characterizing consecutive extremal strings

As the automaton guesses a sequence of extremal strings as in Proposition 10.33, it will place pebbles from an extremal string to a consecutive one. In addition to verifying consecutiveness, this requires the automaton to know which positions in consecutive extremal strings match, in the sense that they correspond to the same position in the input w. When we (conceptually) add a new data value to the input positions, the positions in the extremal string to which elements of the data word are mapped may shift around or disappear completely. For example, a new $\Gamma_{1\mathrm{top}}$ letter may pop up between already-existing Γ_{rest} letters, causing their position in the current extremal string to have shifted w.r.t. the previous extremal string. Of course, the positions still correspond to the same elements as before, and we would like to keep track of this.

Verifying consecutiveness and keeping track of matching positions is easy if we have the underlying task word as a sort of common ground. Indeed, for a given task word $\mathcal{T}$ and an extremal string $s' = \mathsf{ext}(\mathcal{T})$, there is a bijective mapping from the *extremal elements* of $\mathcal{T}$ that s' stores, to their positions in s'. The same holds for $\mathcal{T}^{\setminus\!\setminus 1}$ and $s = \mathsf{ext}(\mathcal{T}^{\setminus\!\setminus 1})$. So to get a (partial) mapping from the positions of s to the positions of s' such that the element of $\mathcal{T}$ that is referred to is preserved, we can inverse $\mathsf{Emb}_{\mathsf{ext}\circ\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}}$ and compose it with $\mathsf{Emb}_{\mathsf{ext}\circ\mathsf{abst},\mathcal{T}}$. This might be a *partial* mapping since some positions may no longer be extremal once the abstracted data values are updated. This *partial embedding from s to s' via $\mathcal{T}$* which keeps track of the matching positions will be a stepping stone towards eliminating $\mathcal{T}$ altogether.

To make this precise, we define the extremal elements of a task word:

Definition 10.34 ($\mathsf{ExtElem}(\mathcal{T})$)**.** *For a $\mathcal{D}$-tasked word $\mathcal{T}$, let*

$$\mathsf{ExtElem}(\mathcal{T}) = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\mathsf{ExtPos}(\mathsf{abst}(\mathcal{T}))).$$

That is, those elements of $\mathcal{T}$ who get projected in the data abstraction to a position which is extremal. We denote $\mathsf{ExtElem}_{h,\theta}(\mathcal{T})$ and $\mathsf{ExtElem}_{\theta}(\mathcal{T})$ similarly.

Additionally, we describe the relationship between extremal elements of a task word to the extremal elements of its trimming:

Lemma 10.35. *Let $\mathcal{T}$ be a $\mathcal{D}$-task word with universe T. For every $\theta \in \Theta_\exists$:*

1. $\mathsf{ExtElem}_{2\mathrm{top},\theta}(\mathcal{T}) = \mathsf{ExtElem}_{1\mathrm{top},\theta}(\mathcal{T}^{\setminus 1})$
2. $\mathsf{ExtElem}_{\theta}(\mathcal{T}) \subseteq \mathsf{ExtElem}_{2\mathrm{top},\theta}(\mathcal{T}^{\setminus 1}) \cup \mathsf{ExtElem}_{\theta}(\mathcal{T}^{\setminus 1})$
3. $\mathsf{ExtElem}_{\mathrm{rest},\theta}(\mathcal{T}) \subseteq \mathsf{ExtElem}_{2\mathrm{top},\theta}(\mathcal{T}^{\setminus 1}) \cup \mathsf{ExtElem}_{\mathrm{rest},\theta}(\mathcal{T}^{\setminus 1})$

The proof of this lemma is straightforward, but is rather technical since we go back and forth between data words, task words, and their trimmings and abstractions.

Proof. Let $w = \mathsf{abst}(\mathcal{T})$ and $w' = \mathsf{abst}(\mathcal{T}^{\setminus 1})$.

1. Let $d \in D$. Since $value_{\mathcal{T}}(d) = value_{\mathcal{T}^{\setminus 1}}(d) + 1$, We have $w(\mathsf{Emb}^{-1}_{\mathsf{abst},\mathcal{T}})(d) \in \Gamma_{2\mathrm{top}}$ if and only if $w'(\mathsf{Emb}^{-1}_{\mathsf{abst},\mathcal{T}^{\setminus 1}})(d)) \in \Gamma_{1\mathrm{top}}$. Using Lemma 10.8, we have for every $\theta \in \Theta_\exists$ that

   $$\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\mathsf{Pos}_{2\mathrm{top},\theta}(w)) = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus 1}}(\mathsf{Pos}_{1\mathrm{top},\theta}(w')).$$

   Since $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}$ and $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus 1}}$ are order-preserving, for both functions $opt = \max$ and $opt = \min$ we have that the positions

   $$\ell_{opt} = opt(\mathsf{Pos}_{2\mathrm{top},\theta}(w)), \qquad \ell'_{opt} = opt(\mathsf{Pos}_{1\mathrm{top},\theta}(w'))$$

   are obtained under abst from the same $\mathcal{T}$ element

   $$d_{opt} = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\ell_{opt}) = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus 1}}(\ell'_{opt}).$$

2. Let $\ell \in \mathsf{ExtPos}_\theta(w)$, let $d = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\ell)$ be its corresponding element, and let $w(\ell) = (\mathrm{rest}, ts_d)$. Recall that ExtPos_θ only contains promised tasks, so we have that $P_\theta \in ts_d$. Let ℓ' the position to which d is mapped when abstracting $\mathcal{T}^{\setminus 1}$, i.e. $d = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus 1}}(\ell')$, and let $w'(\ell') = (h, ts'_d)$ with $h \in \{2\mathrm{top}, \mathrm{rest}\}$ since this is the abstraction of a trimming.

   By Lemma 10.8, we have $\theta \in \omega(ts'_d)$. Let D' be the universe of $\mathcal{D}^{\setminus 1}$. Since $\mathcal{D}^{\setminus 1}$ is a substructure of $\mathcal{D}$ and $d \in D'$, if $\mathcal{D} \not\models \exists y\ \theta(d, y)$ then also $\mathcal{D}^{\setminus 1} \not\models \exists y\ \theta(d, y)$ and

therefore also $P_\theta \in ts'_d$. Now to continue the proof, assume $\theta(x,y) \models x \leq_1 y$ (the case of $\theta(x,y) \models y \leq_1 x$ is analogous).

Assume for contradiction that both $\ell' \notin \mathsf{ExtPos}_{2\text{top},\theta}(w')$ and $\ell' \notin \mathsf{ExtPos}_\theta(w')$. Since we already established that θ is promised, this implies ℓ' does not correspond to an extremal element and therefore there exists some other extremal element $d_1 \in D'$, who is mapped to position $\ell'_1 \in [|D'|]$ in the abstraction of $\mathcal{T}^{\setminus\!\setminus 1}$, and its layer is $h_1 \in \{2\text{top}, \text{rest}\}$. That is, $d_1 = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}}(\ell'_1)$, $w'(\ell'_1) \in \Gamma_{h_1}$, $\ell' < \ell'_1$, $\mathcal{D} \models d <_1 d_1$, and:

a) if $w'(\ell') \in \Gamma_{2\text{top}}$, then $h_1 = 2\text{top}$, $\ell'_1 \in \mathsf{ExtPos}_{2\text{top},\theta}(w')$, and $\mathcal{D} \models d \sim_2 d_1$;

b) if $w'(\ell') \in \Gamma_{\text{rest}}$, then $h_1 = \text{rest}$, $\ell'_1 \in \mathsf{ExtPos}_\theta(w')$ and $w'(\ell'_1) \in \Gamma_{\text{rest}}$.

Now let us shift our attention to where this element d_1 is mapped in the abstraction of $\mathcal{T}$. Let ℓ_1 be such that $d_1 = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\ell_1)$. We have $\ell < \ell_1$ and $w(\ell_1) \in \Gamma_{\text{rest}}$, since previously it was at most the second-largest data value. Let $w(\ell_1) = (\text{rest}, ts_{d_1})$ and $w'(\ell'_1) = (h_1, ts'_{d_1})$. Since $\ell'_1 \in \mathsf{ExtPos}_{2\text{top},\theta}(w') \cup \mathsf{ExtPos}_\theta(w')$, we have $\theta \in \omega(ts'_{d_1})$. By Lemma 10.8 we have $\theta \in \omega(ts_{d_1})$, and using that $\ell \in \mathsf{ExtPos}_\theta(w)$ we have $C_\theta \in ts_{d_1}$ and hence $\mathcal{D} \models \exists y\ \theta(d_1, y)$. Since $\theta \in \omega(ts_d) \cap \omega(ts_{d_1})$, we have $\xi^{\omega(ts_d)} = \xi^{\omega(ts_{d_1})}$, implying that d and d_1 have the same 1-type in $\mathcal{D}$.

We split into the cases of the layer:

a) If $w'(\ell') \in \Gamma_{2\text{top}}$, we have that $\mathcal{D} \models d \leq_1 d_1 \wedge d \sim_2 d_1$. Since they have the same 1-type, we have that for any element d'' such that $\mathcal{D} \models d_1 \leq_1 d''$, the 2-type of (d_1, d'') is the same as the 2-type of (d, d''). Since $\theta(x,y) \models x \leq_1 y$ by our assumption, and since $\mathcal{D} \models \exists y\ \theta(d_1, y)$ there exists some $d'' \geq_1 d_1$ such that $\mathcal{D} \models \theta(d_1, d'')$. Therefore also $\mathcal{D} \models \theta(d, d'')$ and $\mathcal{D} \models \exists y\ \theta(d, y)$, in contradiction to $P_\theta \in ts_d$.

b) If $w'(\ell') \in \Gamma_{\text{rest}}$, since $\ell'_1 \in \mathsf{ExtPos}_\theta(w')$ we have by definition that $P_\theta \in ts'_{d_1}$ and hence $\mathcal{D}^{\setminus 1} \not\models \exists y\ \theta(d_1, y)$. But we also have $\mathcal{D} \models \exists y\ \theta(d_1, y)$, denote by d'' the element such that $\mathcal{D} \models \theta(d_1, d'')$. Since $\mathcal{D}^{\setminus 1} \not\models \exists y\ \theta(d_1, y)$, we conclude that $value_{\mathcal{D}}(d'') = maxval_{\mathcal{D}}$. Since both and d and d_1 have data value at most $maxval_{\mathcal{D}} - 2$, it holds that $\mathcal{D} \models d \not\sim_2 d'' \wedge \neg S_2(d, d'')$ and $\mathcal{D} \models d_1 \not\sim_2 d'' \wedge \neg S_2(d_1, d'')$. Since $\mathcal{D} \models d \leq_1 d_1$ and $\theta(x,y) \models x \leq_1 y$, also $\mathcal{D} \models d \leq_1 d''$, and all in all, the 2-type of (d_1, d'') is the same the 2-type of (d, d''). Therefore, $\mathcal{D} \models \theta(d, d'')$, implying $\mathcal{D} \models \exists y\ \theta(d, y)$, in contradiction to $P_\theta \in ts_d$.

3. For every $d \in D$, $value_{\mathcal{D}}(d) = value_{\mathcal{D}^{\setminus 1}}(d) - 1$. Hence, using Lemma 10.8, we have that for every $\theta \in \Theta_\exists$, the embedding $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\mathsf{Pos}_{\text{rest},\theta}(w)))$ is given by

$$\mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}}(\mathsf{Pos}_{2\text{top},\theta}(w') \cup \mathsf{Pos}_{\text{rest},\theta}(w')).$$

Since $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}$ and $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}}$ are order-preserving, for both functions $opt = \max$ and $opt = \min$ we have that the positions $\ell'_{opt} = opt(\mathsf{Pos}_{2\text{top},\theta}(w') \cup \mathsf{Pos}_{\text{rest},\theta}(w'))$

and $\ell_{opt} = opt(\mathsf{Pos}_{\mathrm{rest},\theta}(w)))$ are obtained under abst from the same $\mathcal{T}$ element $d_{opt} = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\ell_{opt}) = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}}(\ell'_{opt})$.

□

All of this work was in order to ensure that the following is well-defined. Let (s', s) be a pair of consecutive extremal strings, and let $\mathcal{T}$ be a task word such that $s = \mathsf{ext}(\mathcal{T})$ and $s' = \mathsf{ext}(\mathcal{T}^{\setminus\!\setminus 1})$. We denote by $\mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}$ the function from the set of positions ℓ for which $s(\ell) \notin \Gamma_{1\mathrm{top}}$ to $[|s'|]$ defined as follows:

$$\mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}(\ell) = \mathsf{Emb}^{-1}_{\mathsf{ext}\circ\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}}\left(\mathsf{Emb}_{\mathsf{ext}\circ\mathsf{abst},\mathcal{T}}(\ell)\right).$$

$\mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}$ is well-defined: since $s(\ell) \notin \Gamma_{1\mathrm{top}}$, we have from Lemma 10.35 that

$$\mathsf{Emb}_{\mathsf{ext}\circ\mathsf{abst},\mathcal{T}}(\ell) \in \mathsf{Emb}_{\mathsf{ext}\circ\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}}([|s'|]).$$

We refer to $\mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}$ as a *partial embedding via* $\mathcal{T}$ since it is injective and order-preserving.

Example 10.36. *The domain of the partial embedding* $\mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}$ *for* s, s', *and* $\mathcal{T}$ *from Example 10.17 is* $\{1, 2, 4\}$, *and* $\mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}$ *is the composition:*

$$\mathsf{Emb}^{-1}_{\mathsf{ext},\mathsf{abst}(\mathcal{T}^{\setminus\!\setminus 1})} \circ \mathsf{Emb}^{-1}_{\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}} \circ \mathsf{Emb}_{\mathsf{abst},\mathcal{T}} \circ \mathsf{Emb}_{\mathsf{ext},\mathsf{abst}(\mathcal{T})}$$

The embeddings $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}^{\setminus\!\setminus 1}}$ *and* $\mathsf{Emb}_{\mathsf{abst},\mathcal{T}}$ *were given in Example 10.15, and the embeddings* $\mathsf{Emb}_{\mathsf{ext},\mathsf{abst}(\mathcal{T}^{\setminus\!\setminus 1})}$ *and* $\mathsf{Emb}_{\mathsf{ext},\mathsf{abst}(\mathcal{T})}$ *were given in Example 10.17. The partial embedding is given by:*

$$\mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}(1) = 1, \quad \mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}(2) = 3, \quad \mathsf{PEmb}^{\mathcal{T}}_{s\hookrightarrow s'}(4) = 4$$

To define partial embeddings *independently* of $\mathcal{T}$, we show that the specific task word witnessing the consecutiveness of extremal strings is immaterial:

Lemma 10.37. *Let* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *be task words, and let*

$$\begin{array}{rcccc} s & = & \mathsf{ext}(\mathsf{abst}(\mathcal{T}_1)) & = & \mathsf{ext}(\mathsf{abst}(\mathcal{T}_2)) \\ s' & = & \mathsf{ext}(\mathsf{abst}(\mathcal{T}_1^{\setminus\!\setminus 1})) & = & \mathsf{ext}(\mathsf{abst}(\mathcal{T}_2^{\setminus\!\setminus 1})) \end{array}$$

Then $\mathsf{PEmb}^{\mathcal{T}_1}_{s\hookrightarrow s'} = \mathsf{PEmb}^{\mathcal{T}_2}_{s\hookrightarrow s'}$.

Proof. Since $\mathsf{ext}(\mathcal{T}_1) = \mathsf{ext}(\mathcal{T}_2)$, the domains of $\mathsf{PEmb}^{\mathcal{T}_1}_{s\hookrightarrow s'}$ and $\mathsf{PEmb}^{\mathcal{T}_2}_{s\hookrightarrow s'}$ are equal. Assume for contradiction that $\mathsf{PEmb}^{\mathcal{T}_1}_{s\hookrightarrow s'} \neq \mathsf{PEmb}^{\mathcal{T}_2}_{s\hookrightarrow s'}$. Let ℓ_s be the left-most (that is, smallest in the linear order) position in s such that $s(\ell_s) \notin \Gamma_{1\mathrm{top}}$ and $\mathsf{PEmb}^{\mathcal{T}_1}_{s\hookrightarrow s'}(\ell_s) \neq$

Table 10.1

In each row there is one element d of $\mathcal{T}_i^{\Vdash 1}$ with $i \in \{1,2\}$. Each of the columns $u \in \{w_1, w_2, s\}$ indicates the position of d in the string u according to the embedding $\mathsf{Emb}_{Op,\mathcal{T}_i}$ with the appropriate operation $Op = \mathsf{abst}$ for $u \in \{w_1, w_2\}$ and $Op = \mathsf{ext} \circ \mathsf{abst}$ for $u = s$. Each of the columns $u' \in \{s, w_2'\}$ indicates the position of d in the string u' according to the embedding $\mathsf{Emb}_{Op,\mathcal{T}_i^{\Vdash 1}}$ with $Op = \mathsf{abst}$ for $u' = w_2'$ and $Op = \mathsf{ext} \circ \mathsf{abst}$ for $u' = s$. Each of the elements and positions in row 1 (respectively row 3) are strictly smaller than the elements and positions in the same column in row 2 (respectively row 4). (The comparison of elements is with respect to $\leq_1$.)

	$\mathcal{T}_1^{\Vdash 1}$	$\mathcal{T}_2^{\Vdash 1}$	w_1	w_2	s	s'	w_2'
smaller		$d_{1,2}$		$\ell_{1,2}$		$\ell_{s',1}$	$\ell_{w_2',1}$
larger		d_2		ℓ_2	ℓ_s	$\ell_{s',2}$	
smaller	d_1		ℓ_1		ℓ_s	$\ell_{s',1}$	
larger	$d_{2,1}$		$\ell_{2,1}$			$\ell_{s',2}$	

$\mathsf{PEmb}^{\mathcal{T}_2}_{s \hookrightarrow s'}(\ell_s)$. For $i = 1, 2$, let $w_i = \mathsf{abst}(\mathcal{T}_i)$, $n_i = |w_i|$, and $w_i' = \mathsf{abst}(\mathcal{T}_i^{\Vdash 1})$. Let ℓ_i, d_i, $\ell_{s',i}$ be as follows:

$$\begin{aligned} \ell_i &= \mathsf{Emb}_{\mathsf{ext},w_i}(\ell_s) \\ d_i &= \mathsf{Emb}_{\mathsf{abst},\mathcal{T}_i}(\ell_i) \\ \ell_{s',i} &= \mathsf{PEmb}^{\mathcal{T}_i}_{s \hookrightarrow s'}(\ell_s) \end{aligned}$$

and note $d_i = \mathsf{Emb}_{\mathsf{ext} \circ \mathsf{abst},\mathcal{T}_i}(\ell_s) = \mathsf{Emb}_{\mathsf{ext} \circ \mathsf{abst},\mathcal{T}_i^{\Vdash 1}}(\ell_{s',i})$, and since $s(\ell_s) \notin \Gamma_{1\mathrm{top}}$, d_i belongs to the universe of $\mathcal{T}_i^{\Vdash 1}$. For distinct $i, j \in \{1,2\}$, let $d_{i,j}$, $\ell_{i,j}$, $\ell_{w_2',1}$ be as follows:

$$\begin{aligned} \ell_{w_j',i} &= \mathsf{Emb}_{\mathsf{ext},w_j'}(\ell_{s',i}) \\ d_{i,j} &= \mathsf{Emb}_{\mathsf{abst},\mathcal{T}_j^{\Vdash 1}}(\ell_{w_j',i}) \\ \ell_{i,j} &= \left(\mathsf{Emb}_{\mathsf{abst},\mathcal{T}_j}\right)^{-1}(d_{i,j}) \end{aligned}$$

and note $d_{i,j} = \mathsf{Emb}_{\mathsf{ext} \circ \mathsf{abst},\mathcal{T}_j^{\Vdash 1}}(\ell_{s',i})$, and that $d_{i,j}$ belongs to the universe of $\mathcal{T}_j^{\Vdash 1}$ and hence to that of $\mathcal{T}_j$. W.l.o.g. assume that $\ell_{s',1} < \ell_{s',2}$, and therefore by the order-preservation property of embeddings, we have $\mathcal{T}_1 \models d_1 <_1 d_{2,1}$, $\ell_1 < \ell_{2,1}$, $\mathcal{T}_2 \models d_{1,2} <_1 d_2$, and $\ell_{1,2} < \ell_2$ (see Table 10.1).

Let

$$\begin{aligned} ts(s(\ell_s)) &= ts_s \\ ts(w_i(\ell_i)) &= ts_i \\ ts(s'(\ell_{s',i})) &= ts_{s',i} \\ ts(w_j(\ell_{i,j})) &= ts_{i,j} \end{aligned}$$

From the definition of ext, $ts_s = ts_1 = ts_2$. By the definitions ext and abst and from Lemma 10.8, ts_s, $ts_{s',1}$, $ts_{s',2}$, $ts_{1,2}$, and $ts_{2,1}$ all realize the same set-type.

Before continuing the proof of Lemma 10.37, we prove several claims. They are all pretty straightforward, but we phrase them as claims nonetheless in order to not obfuscate the proof with their short justifications.

Claim 10.38. *Let (s', s) be a pair of consecutive extremal strings, and let $\mathcal{T}$ be a task word such that $s = \mathsf{ext}(\mathcal{T})$ and $s' = \mathsf{ext}(\mathcal{T}^{\setminus 1})$. Let $\ell \in [|s|]$. Then:*

1. *$s(\ell) \in \Gamma_{2\text{top}}$ if and only if $s'(\mathsf{PEmb}^{\mathcal{T}}_{s \hookrightarrow s'}(\ell)) \in \Gamma_{1\text{top}}$.*
2. *$s(\ell) \in \Gamma_{\text{rest}}$ if and only if $s'(\mathsf{PEmb}^{\mathcal{T}}_{s \hookrightarrow s'}(\ell)) \in \Gamma_{2\text{top}} \cup \Gamma_{\text{rest}}$.*

Proof. The claim follows from the definitions of $\mathsf{PEmb}^{\mathcal{T}}_{s \hookrightarrow s'}$, ext, and abst, and from Lemma 10.35. □

Claim 10.39. $\ell_{1,2} \notin \mathsf{ExtPos}(w_2)$.

Proof. Assume for contradiction that $\ell_{1,2} \in \mathsf{ExtPos}(w_2)$. Then there exists $\tilde{\ell}_{1,2} \in [|s|]$ such that $\tilde{\ell}_{1,2} = \mathsf{Emb}_{\mathsf{ext},w_2}(\ell_{1,2})$ and $d_{1,2} = \mathsf{Emb}_{\mathsf{ext} \circ \mathsf{abst},\mathcal{T}_2}(\tilde{\ell}_{1,2})$ and we have that $\mathsf{PEmb}^{\mathcal{T}_2}_{s \hookrightarrow s'}(\tilde{\ell}_{1,2}) = \ell_{s',1}$. Since $d_{1,2} <_1 d_2$ we have $s(\tilde{\ell}_{1,2}) \notin \Gamma_{1\text{top}}$ and $\tilde{\ell}_{1,2} < \ell_s$. Since $\mathsf{PEmb}^{\mathcal{T}_1}_{s \hookrightarrow s'}$ is injective and $\mathsf{PEmb}^{\mathcal{T}_1}_{s \hookrightarrow s'}(\ell_s) = \ell_{s',1}$, we must have $\mathsf{PEmb}^{\mathcal{T}_1}_{s \hookrightarrow s'}(\tilde{\ell}_{1,2}) \neq \ell_{s',1}$, in contradiction to the minimality of ℓ_s. □

Claim 10.40. *Let $\mathcal{T}$ be a $\mathcal{D}$-task word and let d, d' be elements such that $\mathcal{D} \models d \leq_1 d'$, $value_{\mathcal{D}}(d) \leq maxval_{\mathcal{D}} - 2$, and $value_{\mathcal{D}}(d') \leq maxval_{\mathcal{D}} - 2$. Let $\mathcal{T}^{\setminus 1} \models ts_1(d)$ and $\mathcal{T}^{\setminus 1} \models ts'_1(d')$ such that $\omega(ts_1) = \omega(ts'_1)$. Let $\theta \in \Theta_\exists$ such that $P_\theta \in ts_1 \cap ts'_1$. Finally, let $\mathcal{T} \models ts(d)$ and $\mathcal{T} \models ts'(d')$.*

1. *If $\theta(x, y) \models x \leq_1 y$ and $C_\theta \in ts'$, then $C_\theta \in ts$.*
2. *If $\theta(x, y) \models y \leq_1 x$ and $C_\theta \in ts$, then $C_\theta \in ts'$.*

Proof. Since ts_1 and ts'_1 realize the same witness type set, d and d' have the same 1-type in $\mathcal{D}$. Since $P_\theta \in ts_1 \cap ts'_1$, we have $\mathcal{D}^{\setminus 1} \not\models \exists y\ \theta(d, y)$ and $\mathcal{D}^{\setminus 1} \not\models \exists y\ \theta(d', y)$. Then the claim follows nearly identical arguments to the ones used in the proof of the second case of Lemma 10.35 and the definition of task words.

□

Claim 10.41. *Let* $P_{\bar\theta} \in ts_s$, $s(\ell_s) \in \Gamma_{\text{rest}}$, *and* $\ell_2 \in \mathsf{ExtPos}_{\bar\theta}(w_2)$. *Then* $\bar\theta(x,y) \models x \leq_1 y$ *and* $P_{\bar\theta} \in ts_{2,1}$.

Proof. Since $s(\ell_s) \in \Gamma_{\text{rest}}$, by Claim 10.38 we have $s'(\ell_{s',1}), s'(\ell_{s',2}) \in \Gamma_{2\text{top}} \cup \Gamma_{\text{rest}}$. Hence

$$value_{\mathcal{D}}(d_1), value_{\mathcal{D}}(d_2), value_{\mathcal{D}}(d_{2,1}), value_{\mathcal{D}}(d_{1,2}) \leq maxval_{\mathcal{D}} - 2 = maxval_{\mathcal{D}^{\setminus 1}} - 1$$

and $s(\ell_{1,2}), s(\ell_{2,1}) \in \Gamma_{\text{rest}}$. Since $\omega(ts_s) = \omega(ts_{1,2}) = \omega(ts_{2,1})$, we have $\bar\theta \in \omega(ts_{1,2}) \cap \omega(ts_{2,1})$. Since $\mathcal{D}^{\setminus 1}$ is a substructure of $\mathcal{D}$ and $P_{\bar\theta} \in ts_s$, we also have $P_{\bar\theta} \in ts_{s',1} \cap ts_{s',2}$.

Assume for contradiction that $\bar\theta(x,y) \models y <_1 x$. If $P_{\bar\theta} \in ts_{1,2}$, then since $\ell_{1,2} < \ell_2$, we have

$$\ell_2 \notin \mathsf{ExtPos}_{\bar\theta}(w_2) = \{\ell \mid \ell = \min(\mathsf{Pos}_{\text{rest},P_\theta}(w))\}$$

in contradiction. If $C_{\bar\theta} \in ts_{1,2}$, then since $P_{\bar\theta} \in ts_{s',1} \cap ts_{s',2}$, and since $\bar\theta(x,y) \models y <_1 x$, from Claim 10.40 (with $d = d_{1,2}$ and $d' = d_2$) it follows that $C_{\bar\theta} \in ts_s$, in contradiction to $P_{\bar\theta} \in ts_s$. Hence $\bar\theta(x,y) \models x \leq_1 y$.

Since $\bar\theta \in \omega(ts_{2,1})$, either $P_{\bar\theta} \in ts_{2,1}$ or $C_{\bar\theta} \in ts_{2,1}$. If $C_{\bar\theta} \in ts_{2,1}$, then from Claim 10.40 (with $d = d_1$ and $d' = d_{2,1}$) it follows that $C_{\bar\theta} \in ts_s$, in contradiction to $P_{\bar\theta} \in ts_s$. hence $P_{\bar\theta} \in ts_{2,1}$. □

We are now ready to resume the proof of Lemma 10.37. We divide into cases depending on whether $s(\ell_s) \in \Gamma_{2\text{top}}$ or $s(\ell_s) \in \Gamma_{\text{rest}}$.

- First assume $s(\ell_s) \in \Gamma_{2\text{top}}$. Since $\ell_{s',1} = \mathsf{PEmb}^{\mathcal{T}_1}_{s \hookrightarrow s'}(\ell_s)$, we have $s'(\ell_{s',1}) \in \Gamma_{1\text{top}}$ by Claim 10.38. Since $s' = \mathsf{ext}(w_2')$, we have that $\ell_{w_2',i} \in \mathsf{ExtPos}(w_2')$, and hence there is some $\theta \in \omega(ts_s)$ such that $\ell_{w_2',i} \in \mathsf{ExtPos}_{1\text{top},\theta}(w_2')$ and $d_{1,2} \in \mathsf{Emb}_{\mathsf{abst},\mathcal{T}_2^{\setminus\setminus 1}}(\mathsf{ExtPos}_{1\text{top},\theta}(w_2'))$. Hence by Lemma 10.35(1) we have that $d_{1,2} \in \mathsf{Emb}_{\mathsf{abst},\mathcal{T}_2}(\mathsf{ExtPos}_{2\text{top},\theta}(w_2))$, which implies that $\ell_{1,2} \in \mathsf{ExtPos}(w_2)$, in contradiction to Claim 10.39.

- Now assume $s(\ell_s) \in \Gamma_{\text{rest}}$. We have $s'(\ell_{s',1}), s'(\ell_{s',2}) \in \Gamma_{2\text{top}} \cup \Gamma_{\text{rest}}$ by Claim 10.38, i.e.

  $$d_{1,2}, d_{2,1} \leq maxval_{\mathcal{D}^{\setminus 1}} - 1 = maxval_{\mathcal{D}} - 2$$

  Hence, $s(\ell_{1,2}), s(\ell_{2,1}) \in \Gamma_{\text{rest}}$. Since $\ell_2 = \mathsf{Emb}_{\mathsf{ext},w_2}(\ell_s)$ we have that $\ell_2 \in \mathsf{ExtPos}(w_2)$. Let $\theta \in \omega(ts_s) = \omega(ts_{1,2})$. By Claim 10.39 and using that $s(\ell_{1,2}) \in \Gamma_{\text{rest}}$, there exists $\tilde\ell_{1,2} < \ell_{1,2}$ such that $\tilde\ell_{1,2} \in \mathsf{ExtPos}_{\text{rest},\theta}(w_2)$, so let $\ell_{0,\theta}$ be such that $\tilde\ell_{1,2} = \mathsf{Emb}_{\mathsf{ext},w_2}(\ell_{0,\theta})$. We have that $\tilde\ell_{1,2} < \ell_2$, and hence $\ell_{0,\theta} < \ell_s$. Let $s(\ell_{0,\theta}) = (h_{0,\theta}, ts_{0,\theta})$. Since $\tilde\ell_{1,2} \in \mathsf{ExtPos}_{\text{rest},\theta}(w_2)$ we have $\theta \in \omega(ts_{0,\theta})$. Let $\ell_{3,\theta}$ be such that $\ell_{2,1} = \mathsf{Emb}_{\mathsf{ext},w_1}(\ell_{3,\theta})$. Since $\ell_1 < \ell_{2,1}$ we have $\ell_s < \ell_{3,\theta}$. We have $\theta \in \omega(\ell_{2,1}) = \omega(\ell_{3,\theta})$. Hence, $\ell_2 \notin \mathsf{ExtPos}_{\text{rest},\theta}(w_2)$ for all $\theta \in \omega(ts)$. Consequently, there is $\bar\theta$ such that $\ell_2 \in \mathsf{ExtPos}_{\bar\theta}(w_2)$ and $P_{\bar\theta} \in ts_2 = ts_s$.

By Claim 10.41 we have $\bar{\theta}(x,y) \models x \leq_1 y$ and $P_{\bar{\theta}} \in ts_{2,1}$, and hence there is $\ell_{\bar{\theta},w_1}$ such that $\mathsf{ExtPos}_{\bar{\theta}}(w_1) = \{\ell_{\bar{\theta},w_1}\}$ and $\ell_{\bar{\theta},w_1} = \max(\mathsf{Pos}_{\mathrm{rest},P_{\bar{\theta}}}(w_1))$. We have $P_{\bar{\theta}} \in ts(w_1(\ell_{\bar{\theta},w_1}))$ and $\ell_1 < \ell_{2,1} \leq \ell_{\bar{\theta},w_1}$. Let $\ell_{\bar{\theta},s} \in [|s|]$ and $\ell_{\bar{\theta},w_2} \in [|w_2|]$ be such that

$$\ell_{\bar{\theta},w_1} = \mathsf{Emb}_{\mathsf{ext},w_1}(\ell_{\bar{\theta},s}), \quad \ell_{\bar{\theta},w_2} = \mathsf{Emb}_{\mathsf{ext},w_2}(\ell_{\bar{\theta},s})$$

Then

$$P_{\bar{\theta}} \in ts(s(\ell_{\bar{\theta},s})) = ts(w_1(\ell_{\bar{\theta},w_1})) = ts(w_2(\ell_{\bar{\theta},w_2})), \quad \ell_s < \ell_{\bar{\theta},s}, \quad \ell_2 < \ell_{\bar{\theta},w_2}$$

Since $w_1(\ell_{\bar{\theta},w_1}) \in \Gamma_{\mathrm{rest}}$ and $w_1(\ell_{\bar{\theta},w_1}) = s(\ell_{\bar{\theta},s}) = w_2(\ell_{\bar{\theta},w_2})$, we have $\ell_2 \notin \mathsf{ExtPos}_{\bar{\theta}}(w_2) = \{\max(\mathsf{Pos}_{\mathrm{rest},P_{\bar{\theta}}}(w_2))\}$, in contradiction.

□

At long last, we can define the partial embedding between the positions of consecutive extremal strings independently of any *specific* task word that witnesses their consecutiveness:

Definition 10.42 (Partial embedding)**.** *Let (s', s) be consecutive extremal strings. We denote by $\mathsf{PEmb}_{s \hookrightarrow s'}$ the function from $s^{-1}(\Gamma - \Gamma_{1\mathrm{top}})$ to $[|s'|]$ defined as follows. Let $\mathcal{T}$ be a task word, and $s = \mathsf{ext}(\mathsf{abst}(\mathcal{T}))$ and $s' = \mathsf{ext}(\mathsf{abst}(\mathcal{T}^{\setminus\!\setminus 1}))$. Then $\mathsf{PEmb}_{s \hookrightarrow s'} = \mathsf{PEmb}^{\mathcal{T}}_{s \hookrightarrow s'}$. This is well-defined by Lemma 10.37.*

But as we stand, we are still requiring *some* task word to witness consecutiveness. Next, we get rid of the background task word altogether, by showing that if s, s' are consecutive, then s' can be obtained by guessing a *string* r that will get new data values, interleaving it into the proper positions g of s, which can also be guessed, and updating the abstracted data values. We also show that the partial embedding that keeps track of matching the positions can be obtained using r and g only. Since extremal strings have bounded length, it will follow that these r also have bounded length.

Word of caution: the next section is technical by nature, and the definition of the automaton $\mathcal{A}^{\varphi}$ is perfectly understandable given the groundwork so far. The reason to eliminate the task words is so we can later prove that $\mathcal{A}^{\varphi}$ can be *constructed* without concrete task words (and to bound the complexity of said construction), which is what Lemma 10.46 will be used for.

10.3.1 A syntactic representation of consecutive extremal strings

The extremal string $r \downarrow_g s^0$ simulates the extension of a task word $\mathcal{T}^0$ whose extremal string is s^0 by adding elements with a new maximal data value. The letters of these elements are determined by r and their placement in the linear order of $\mathcal{T}^0$ is determined by g.

Definition 10.43 ($r \downarrow_g s^0$)**.** *Let $s \in \Gamma^\star$. We denote by $\downarrow s$ the string that is obtained from s by substituting letters of the form $(1\text{top}, ts)$ with $(2\text{top}, ts)$ and letters of the form $(2\text{top}, ts)$ or (rest, ts) with (rest, ts). Let $s^0 \in \mathsf{EXT}(\Gamma)$, $r \in \Gamma^\star_{1\text{top}} \cap \Gamma^\star_{\text{P}}$, and let $g : [|r|] \to [|r| + |s^0|]$ be a strictly monotone function. Then let $s^1 = r \sqcup_g \downarrow s^0$, that is, s^1 is the g-shuffle of r and $\downarrow s^0$ (see Definition 8.17).*

We now define the string $r \downarrow_g s^0$, which is obtained from s^1 substituting every letter (h, ts) at position $\ell \in [|s^1|]$ with a letter (h, ts'). Essentially, this replacement updates the tasks to take into account the new elements described by r (and whose position is given by g). The letter (h, ts') is such that it realizes the same set-type, i.e. $\omega(ts) = \omega(ts')$ and, for every $\theta \in \omega(ts)$, we have $C_\theta \in ts'$ if θ is completed, that is, if any of the following hold

1. *θ was already completed: $C_\theta \in ts$.*
2. *The necessary witness ℓ_2 for ℓ was found to the right: $\theta \models_{\mathrm{DW}(\Xi)} x \leq_1 y$ and there is $\ell_2 \in [|s^1|]$ such that $\ell \leq \ell_2$, either $\ell \in \mathsf{Pos}_{1\text{top}}(s^1)$ or $\ell_2 \in \mathsf{Pos}_{1\text{top}}(s^1)$, and $\mathrm{perfect}_{s^1(\ell), s^1(\ell_2)}(x, y) \equiv_{\mathrm{DW}(\Xi)} \theta(x, y)$.*
3. *The necessary witness ℓ_2 for ℓ was found to the left: $\theta \models_{\mathrm{DW}(\Xi)} y <_1 x$ and there is $\ell_1 \in [|s^1|]$ such that $\ell_1 < \ell$, either $\ell_1 \in \mathsf{Pos}_{1\text{top}}(s^1)$ or $\ell \in \mathsf{Pos}_{1\text{top}}(s^1)$, and $\mathrm{perfect}_{s^1(\ell_1), s^1(\ell)}(y, x) \equiv_{\mathrm{DW}(\Xi)} \theta(x, y)$.*

Otherwise, since θ was not completed, it remains as a promised task and we have $P_\theta \in ts'$.

If we have the $\mathcal{D}$-task word $\mathcal{T}$ at hand, then we can obtain the string r and the function g for s and s^0. In fact, obtaining r is quite easy: it suffices to look at the elements with maximal data value, and substitute every letter $(1\text{top}, ts)$ with $(1\text{top}, ts_P)$ such that $ts_P = \{P_\theta \mid \theta \in \omega(ts)\}$.

We now prove this lemma which will be useful later. It essentially states that in order to check satisfaction of constraints, it is enough to consider the elements that correspond to extremal positions.

Lemma 10.44 (Extremal witnesses)**.** *Let $\mathcal{T}$ be a $\mathcal{D}$-task word. Let d, d_0 be elements of $\mathcal{T}$ and $\theta \in \Theta_\exists$. If $\mathcal{D} \models \theta(d, d_0)$ and $maxval_\mathcal{D} \in \{value_\mathcal{D}(d), value_\mathcal{D}(d_0)\}$, then there is an element $d' \in \mathsf{ExtElem}(\mathcal{T})$ such that $\mathcal{D} \models \theta(d, d')$.*

Proof. Let $w = \mathsf{abst}(\mathcal{T})$. Let $w(\mathsf{Emb}^{-1}_{\mathsf{abst},\mathcal{T}}(d)) = (h, ts)$, and let $w(\mathsf{Emb}^{-1}_{\mathsf{abst},\mathcal{T}}(d_0)) = (h_0, ts_0)$. Let $\theta \in ts_0$. There are $d_{0,1} \leq_1 d_0 \leq_1 d_{0,2}$ such that

1. $d_{0,1}, d_{0,2} \in \mathsf{ExtElem}_{\theta,h}(\mathcal{T})$,
2. $\mathcal{D} \models \xi^{\omega(ts_0)}(d_{0,1})$ and $\mathcal{D} \models \xi^{\omega(ts_0)}(d_{0,2})$,
3. $value_\mathcal{D}(d_0) = value_\mathcal{D}(d_{0,1}) = value_\mathcal{D}(d_{0,2})$ if $value_\mathcal{D}(d_0) \geq maxval_\mathcal{D} - 1$,

4. both $value_{\mathcal{D}}(d_{0,1}) \leq maxval_{\mathcal{D}} - 2$ and $value_{\mathcal{D}}(d_{0,2}) \leq maxval_{\mathcal{D}} - 2$ if $value_{\mathcal{D}}(d_0) \leq maxval_{\mathcal{D}} - 2$, and

5. $d_0, d_{0,1}, d_{0,2}$ have the same 1-type in $\mathcal{D}$.

Since $maxval_{\mathcal{D}} \in \{value_{\mathcal{D}}(d), value_{\mathcal{D}}(d_0)\}$, the 2-types of (d, d_0) and either $(d, d_{0,1})$ or $(d, d_{0,2})$ are equal depending on whether $\theta(x,y) \models x \leq_1 y$; for the case that $value_{\mathcal{D}}(d_0) \geq maxval_{\mathcal{D}} - 1$ we use that $\mathcal{D} \models d_0 \sim_2 d_{0,1}$ and $\mathcal{D} \models d_0 \sim_2 d_{0,2}$. For the case that $value_{\mathcal{D}}(d_0) \leq maxval_{\mathcal{D}} - 2$, the same argument holds using the fact that $\mathcal{D} \models d_0 \not\sim_2 d \wedge \neg S_2(d_0, d)$. □

Lemma 10.45. *Given a $\mathcal{D}$-task word $\mathcal{T}$ and two extremal strings s and s^0 such that $s = \mathsf{ext}(\mathcal{T})$ and $s^0 = \mathsf{ext}(\mathcal{T}^{\mathbb{1}})$, we can effectively obtain an $r \in \Gamma^\star_{\mathrm{1top}} \cap \Gamma^\star_{\mathrm{P}}$ and $g : [|s^0|] \to [|s^0| + |s|]$ such that $s = \mathsf{ext}(r \downarrow_g s^0)$.*

Proof. We first need some additional notation. Let

$$\begin{aligned} w &= \mathsf{abst}(\mathcal{T}) \\ X' &= \mathsf{ExtElem}(\mathcal{T}^{\mathbb{1}}) \\ X_{\mathrm{1top}} &= \mathsf{Emb}_{\mathsf{abst},\mathcal{T}}(\mathsf{Pos}_{\mathrm{1top}}(w)) \\ X &= X_{\mathrm{1top}} \cup X' \\ w_X &= \mathsf{abst}(\mathcal{T}|_X) \\ w_{X_{\mathrm{1top}}} &= \mathsf{abst}(\mathcal{T}|_{X_{\mathrm{1top}}}) \end{aligned}$$

The string $r \in \Gamma^\star_{\mathrm{1top}} \cap \Gamma^\star_{\mathrm{P}}$ is obtained from $w_{X_{\mathrm{1top}}}$ by substituting every letter $(\mathrm{1top}, ts)$ with $(\mathrm{1top}, ts_P)$ such that $ts_P = \{P_\theta \mid \theta \in \omega(ts)\}$. The function g is given by

$$g = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}|_{X_{\mathrm{1top}}}} \circ \mathsf{Emb}^{-1}_{\mathsf{abst},\mathcal{T}|_X}$$

Finally, let $w' = \mathsf{abst}(\mathcal{T}^{\mathbb{1}})$, and $w_{X'} = \mathsf{abst}(\mathcal{T}|_{X'})$.

By Lemma 10.35, $\mathsf{ExtElem}(\mathcal{T}) \subseteq X$. Hence, $s = \mathsf{ext}(w_X)$. We will prove that $w_X = r \downarrow_g s^0$, and the lemma will follow.

Notice that

$$w_X = w_{X_{\mathrm{1top}}} \sqcup_g w_{X'}, \quad |w_{X_{\mathrm{1top}}}| = |r|, \quad |w_{X'}| = |s^0|$$

and g is strictly monotone as the composition of two order-preserving functions. Hence, s^0, r, and g are as required in the definition of $r \downarrow_g s^0$ (Definition 10.43). Let $\ell \in [|w_X|]$ and $d = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}|_X}(\ell)$. Let $(r \downarrow_g s^0)(\ell) = (h_a, ts_a)$ and $w_X(\ell) = (h_b, ts_b)$. By the construction of s and $r \downarrow_g s^0$, $h_a = h_b$ and $\omega(ts_a) = \omega(ts_b)$. We need to prove that $ts_a = ts_b$. Let s^1 be as in Definition 10.43, $s^1(\ell) = (h_{a^1}, ts^1_a)$, and $\theta \in \omega(ts_a)$.

Assume $C_\theta \in ts_a$. If $C_\theta \in ts_a^1$, then ℓ is not in the image of g, d is an element of $\mathcal{D}^{\setminus 1}$, and there is an element d' of $\mathcal{D}^{\setminus 1}$ such that $\mathcal{D}^{\setminus 1} \models \theta(d, d')$, hence $C_\theta \in ts_b$. Otherwise, there are $\ell_1 < \ell_2 \in [|s^1|]$ such that $\mathrm{perfect}_{s^1(\ell_1),s^1(\ell_2)}(x, y) \equiv_{\mathrm{DW}(\Xi)} \theta(x, y)$ and either $s^1(\ell_1) \in \Gamma_{1\mathrm{top}}$ or $s^1(\ell_2) \in \Gamma_{1\mathrm{top}}$, and either $\theta \models_{\mathrm{DW}(\Xi)} x \leq_1 y$ and $\ell = \ell_1$, or $\theta \models_{\mathrm{DW}(\Xi)} x >_1 y$ and $\ell = \ell_2$. By Lemma 10.26, this implies that $\mathcal{D} \models \exists y\, \theta(d, y)$, and hence $C_\theta \in ts_b$.

Conversely, assume $C_\theta \in ts_b$. There is d' in the universe of $\mathcal{T}$ such that $\mathcal{D} \models \theta(d, d')$. If both d and d' are elements of $\mathcal{D}^{\setminus 1}$, then $C_\theta \in ts_a^1$ and hence $C_\theta \in ts_a$. Otherwise, we have $maxval_{\mathcal{D}} \in \{value_{\mathcal{D}}(d), value_{\mathcal{D}}(d')\}$. By Lemma 10.44, we may assume w.l.o.g. that $d' \in \mathsf{ExtElem}(\mathcal{T})$, and hence $d' \in X$. Let $\ell' \in [|w_X|]$ such that $d' = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}|_X}(\ell')$. Let $\ell_1 < \ell_2 \in [|s^1|]$ be such that $\{\ell, \ell'\} = \{\ell_1, \ell_2\}$. Let $d_i = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}|_X}(\ell_i)$. By Lemma 10.26, $\mathcal{D} \models \mathrm{perfect}_{s^1(\ell_1),s^1(\ell_2)}(d_1, d_2)$. By Observation 10.25, there is a 2-type $\theta'(x, y)$ such that $\mathrm{perfect}_{s^1(\ell_1),s^1(\ell_2)}(x, y) \equiv \theta'(x, y)$. The 2-type $\theta'(x, y)$ is the 2-type of (d_1, d_2). We have $\theta \models x \leq_1 y$ if and only if $\mathcal{D} \models d_1 \leq_1 d_2$. Hence $\theta(x, y) = \theta'(x, y)$ if $\theta \models x \leq_1 y$, and $\theta(x, y) = \theta'(y, x)$ if $\theta \models x >_1 y$. Consequently, $C_\theta \in ts_b$, and overall we get $ts_a = ts_b$.

□

Finally, we are ready to prove that the generation of consecutive extremal strings can be done without a concrete task word:

Lemma 10.46. *Let $\mathcal{T}_0$ be a $\mathcal{D}_0$-task word, $s^0 = \mathsf{ext}(\mathcal{T}_0)$, $r \in \Gamma^\star_{1\mathrm{top}} \cap \Gamma^\star_{\mathrm{P}}$, and let $g : [|r|] \to [|r| + |s^0|]$ be a strictly monotone function. Then s^0 and $\mathsf{ext}(r \downarrow_g s^0)$ are consecutive.*

Proof. Let $s = \mathsf{ext}(r \downarrow_g s^0)$. Without loss of generality, we may assume the universe D_0 of $\mathcal{D}_0$ is disjoint from $\mathbb{N}$. Let $\bar{g}$ be as in the definition of $r \sqcup_g s^0$. Let $\mathcal{D}$ be the data word over Ξ with universe $D_0 \cup [|r|]$ such that:

1. $\mathcal{D}_0$ is the substructure of $\mathcal{D}$ induced by D_0.
2. For every $\ell \in [|r|]$ and $r(\ell) = (h, ts)$, $\mathcal{D} \models \xi^{\omega(ts)}(\ell)$.
3. For every $\ell_1, \ell_2 \in [|r|]$, $\mathcal{D} \models \ell_1 \sim_2 \ell_2$.
4. For every $d \in D_0$ and $\ell \in [|r|]$, $\mathcal{D} \models d <_2 \ell$.
5. For every $\ell_1, \ell_2 \in [|r|]$, $\mathcal{D} \models \ell_1 \leq_1 \ell_2$ if and only if $\ell_1 \leq \ell_2$.
6. For every $d \in D_0$, let $d' \in D_0$ be the maximal element of $\mathsf{ExtElem}(\mathcal{T}_0)$ with respect to $\leq_1$ such that $d' \leq_1 d$, and let $\ell' \in [|s^0|]$ be such that $d' = \mathsf{Emb}_{\mathsf{ext} \circ \mathsf{abst},\mathcal{T}_0}(\ell')$. For every $\ell_1 \in [|r|]$, $\mathcal{D} \models d \leq_1 \ell_1$ if and only if $\bar{g}(\ell') < g(\ell_1)$; if no such d' exists for d then $\mathcal{D} \models d \leq_1 \ell_1$.

Let $\mathcal{T}$ be a $\mathcal{D}$-task word such that, for every $d \in D_0$, there are $ts, ts_0 \in 2^{\mathrm{Tasks}}_\Omega$ such that $\omega(ts) = \omega(ts_0)$, $\mathcal{T} \models ts(d)$, and $\mathcal{T}_0 \models ts_0(d)$. Clearly $\mathsf{ext}(\mathsf{abst}(\mathcal{T}^{\setminus\setminus 1})) = s^0$. Let r, w_X,

$w_{X'}$, and $w_{X_{1\text{top}}}$ be as in Lemma 10.45. By the construction of $\mathcal{D}$,

$$g = \mathsf{Emb}_{\mathsf{abst},\mathcal{T}|_{X_{1\text{top}}}} \circ \mathsf{Emb}^{-1}_{\mathsf{abst},\mathcal{T}|_X}.$$

By Lemma 10.45, $\mathsf{ext}(r \downarrow_g s^0) = \mathsf{ext}(\mathsf{abst}(\mathcal{T}))$, i.e., s^0 and s are consecutive extremal strings.

□

Example 10.47. *Applying* $\downarrow$ *to* s'*, we have*

$$\downarrow s' = (\text{rest}, ts_3^P)(\text{rest}, ts_3^P)(2\text{top}, ts_1^P)(\text{rest}, ts_3^P)$$

Let $r = (1\text{top}, ts_2^P)(1\text{top}, ts_4^P)$*, and let* $g : [2] \to [2+4]$ *be* $g(1) = 4$ *and* $g(2) = 6$*. Then*

$$r \downarrow_g s' = (\text{rest}, ts_3^C)(\text{rest}, ts_3^C)(2\text{top}, ts_1^C)(1\text{top}, ts_2^C)(\text{rest}, ts_3^C)(1\text{top}, ts_4^C).$$

Observe that $\mathsf{ext}(r \downarrow_g s') = s$*, and recall that* (s', s) *are consecutive.*

10.4 Definition of the automaton $\mathcal{A}^\varphi$

We define an automaton $\mathcal{A}^\varphi = (\Xi, m+1, Q, q_{\text{init}}, F, \delta)$ where $Q = Q_e \cup Q_p$ and $m = 7 \cdot |\Theta_\exists|$. $\mathcal{A}^\varphi$ uses one pebble for each existential constraint in $\Theta_\exists$ and each layer in Γ, plus an additional pebble per constraint. It also uses the designated pebble $m+1$ to read non-extremal positions. We describe its states and transitions next, followed by an example computation.

As we mentioned, $\mathcal{A}^\varphi$ runs in iterations, where each iteration is associated with an extremal string. The end of an iteration is noted with a so-called non-prefix state, and we would like the automaton to transition from $(s, \tau) \in Q_e$ representing the extremal string s to $(s', \tau') \in Q_e$ representing a consecutive extremal string s' as it finds the sequence $s_1, \ldots, s_n$ that indicates the input should be accepted.

Non-prefix states. Q_e is the set of states of the form (s, τ) with s a perfect extremal string of length $|s|$ and τ an $(m+1, |s|)$-pebble assignment satisfying the following conditions, which hold when s has just been read in the input:

- ($\mathbf{c}_1$) the pebble $m+1$ was not used to read s, i.e., we have $\tau(m+1) = \bot$,
- ($\mathbf{c}_2$) every position of s has a pebble on it, i.e., if $s \neq \varepsilon$, then $\tau(\ell) \neq \bot$ for all $\ell \in [|s|]$, and if $s \neq \varepsilon$, then $[|s|] \subseteq \tau([m])$, and
- ($\mathbf{c}_3$) if $s = \varepsilon$, there are no pebbles on s, i.e., $\tau(\ell) = \bot$ for all $\ell \in [|s|]$.

Since the automaton can only move one pebble at a time, and verifying that an extremal string is present in the input requires many intermediate steps, we have another set of *prefix states* Q_p which are visited as the automaton processes the prefixes of the current

extremal string. Once the whole extremal string s has been verified to appear in the input, the automaton has the pebbles placed according to τ and it moves to a non-prefix state (s', τ') where τ' indicates how the pebbles are placed if we were looking at the consecutive extremal string s'.

Prefix states. Q_p is the set of states of the forms $(s, \tilde{s}, \tau, 0)$ and $(s, \tilde{s}, \tau, 1)$ for every perfect extremal string s of length $|s|$, non-empty prefix $\tilde{s}$ of s of length $|\tilde{s}|$, and $(m+1, |s|)$-pebble assignment $\tau : [m+1] \to [|s|] \cup \{\bot\}$ satisfying similar conditions as before, but which now hold if only the prefix $\tilde{s}$ has been read:

- ($\mathbf{c}_4$) $\tau(m+1) = \bot$,
- ($\mathbf{c}_5$) $[|\tilde{s}| - 1] \subseteq \tau([m])$, and
- ($\mathbf{c}_6$) pebbles to the right the current prefix had been placed previously, i.e., for every $|\tilde{s}| \leq \ell \leq |s|$, $\ell \in \tau([m])$ if and only if $s(\ell) \notin \Gamma_{1\text{top}}$.

The 0/1 flag in prefix states is used below for deciding where to place the $m+1$ pebble.

Initial state. The initial state is $q_{\text{init}} = (\varepsilon, \rho_\bot) \in Q_e$.

Final states. The final states are $F = \{(s, \tau) \in Q_e \mid s \text{ is completed}\}$.

For a state $q \in Q_p$ of the form $(s, \tilde{s}, \tau, 0)$ or $(s, \tilde{s}, \tau, 1)$, or $q \in Q_e$ of the form (s, τ), denote $\tau_q = \tau$. For a state $q \in Q_p \cup Q_e$, we say a pebble $k \in [m+1]$ is *available in* q if $\tau_q(k) = \bot$. Note that the pebble $m+1$ is available by (c_4) and (c_1). Let $q = (s, \tilde{s}, \tau_q, b)$ or $q = (s, \tau_q)$. Since τ_q is an $(m+1, |s|)$-pebble assignment, there are at most $|s|$ non-available pebbles in q. By the bound on the length of extremal strings, we have $|s| \leq 7 \cdot |\Theta_\exists| < m$, therefore there is at least one pebble $k \in [m]$ which is available in q. All in all, we have:

Observation 10.48. *For every $q \in Q_p \cup Q_e$, there is a pebble $k \in [m]$ such that both pebble k and pebble $m+1$ are available in q.*

We define the transition relation $\delta \subseteq (Q \times Q) \cup (Q \times \text{MOVE}_{m+1} \times \Sigma \times Q)$.

Transitions from prefix states. Let $q = (s, \tilde{s}, \tau_q, b) \in Q_p$.

1. *Non-extremal transitions*: while reading a prefix of a task word, pebble $m+1$ iterates over non-extremal positions. For every letter $\gamma \in \Gamma_{1\text{top}}^{\text{not ext}}(s, |\tilde{s}|)$, we have that $(q, (m+1)\text{-MOVE}_{i,\lhd}, \xi^{\omega(\gamma)}, q') \in \delta$ where $q' = (s, \tilde{s}, \tau_q, 1)$ and

$$i = \begin{cases} \text{argmax}_{0 \leq \ell < |\tilde{s}|} \{t \mid \hat{\tau}_q(t) = \ell\}, & b = 0 \\ m+1, & b = 1 \end{cases}$$

2. *Extremal transitions*: if we are at a new 1-top position, we read it with an available pebble, and if the current position already has a pebble, a silent transition moves on. The automaton will now be on either the next prefix, or the next extremal state if $s = \tilde{s}$, that is, the whole s has been read. Let

$$\tau_{q'} = \begin{cases} \tau_q[k \mapsto |\tilde{s}|], & \tilde{s}(|\tilde{s}|) \in \Gamma_{1\text{top}} \\ \tau_q, & \tilde{s}(|\tilde{s}|) \notin \Gamma_{1\text{top}} \end{cases}$$

and let

$$q' = \begin{cases} (s, \tilde{s}s(|\tilde{s}|+1), \tau_q, 0) & \tilde{s} \neq s \\ (s, \tau_q) & \tilde{s} = s \end{cases}$$

Let $i = \operatorname{argmax}_{0 \leq \ell < |\tilde{s}|}\{t \mid \hat{\tau}_q(t) = \ell\}$ and $j = \operatorname{argmin}_{|\tilde{s}| \leq \ell \leq |s|+1}\{t \mid \hat{\tau}_q(t) = \ell\}$. We have $(q, q') \in \delta$ if $\tilde{s}(|\tilde{s}|) \notin \Gamma_{1\text{top}}$, and $(q, k\text{-MOVE}_{i,j}, \xi^{\omega(\tilde{s}(|\tilde{s}|))}, q') \in \delta$ if $\tilde{s}(|\tilde{s}|) \in \Gamma_{1\text{top}}$ and the pebble k is available in q.

Transitions from non-prefix states. If (s_0, s) are consecutive, we start reading s by moving to $q' = (s, s(1), \tau_{q'}, 0) \in Q_p$, where $\tau_{q'}$ stores the pebble assignment induced by $\mathsf{PEmb}_{s \hookrightarrow s_0}$. For every consecutive pair (s_0, s) of extremal strings and states $q = (s_0, \tau_q) \in Q_e$ and $q' = (s, s(1), \tau_{q'}, 0) \in Q_p$, we have $(q, q') \in \delta$ if $\mathsf{PEmb}_{s \hookrightarrow s_0}$ induces $\tau_{q'}$ as follows: for every pebble $k \in [m]$, $\tau_{q'}(k)$ is $(\mathsf{PEmb}_{s \hookrightarrow s_0})^{-1}(\tau_q(k))$ if $\tau_q(k)$ is in the image of $\mathsf{PEmb}_{s \hookrightarrow s_0}$, and is $\perp$ otherwise.

An accepting computation

Recall φ from Example 9.14:

$$\begin{aligned} \varphi \;=\; & \forall x \forall y\, \chi(x,y) \wedge \\ & \forall x \Big((\xi_1(x) \to \exists y\, (\theta_1(x,y) \vee \theta_3(x,y))) \wedge \\ & \qquad (\xi_2(x) \to \exists y\, (\theta_2(x,y) \vee \theta_4(x,y))) \Big) \end{aligned}$$

Let $u = \xi_1\xi_1\xi_1\xi_2\xi_1\xi_2$. There is an accepting computation of $\mathcal{A}^\varphi$ on u, which in particular induces the data word from Example 9.6. We show the sequence $q_1, \ldots, q_{21}$ of states of this accepting computation. Let $m = 7 \cdot |\Theta_\exists| + 1$. Let $\tau_{\text{I}}, \ldots, \tau_{\text{X}}$ be pebble-assignments:

$$\begin{array}{llll} \tau_{\text{I}}: & [m] \to [1] \cup \{\perp\} & \tau_{\text{VI}}: & [m] \to [4] \cup \{\perp\} \\ \tau_{\text{II}}: & [m] \to [2] \cup \{\perp\} & \tau_{\text{VII}}: & [m] \to [4] \cup \{\perp\} \\ \tau_{\text{III}}: & [m] \to [2] \cup \{\perp\} & \tau_{\text{VIII}}: & [m] \to [5] \cup \{\perp\} \\ \tau_{\text{IV}}: & [m] \to [3] \cup \{\perp\} & \tau_{\text{IX}}: & [m] \to [5] \cup \{\perp\} \\ \tau_{\text{V}}: & [m] \to [3] \cup \{\perp\} & \tau_{\text{X}}: & [m] \to [5] \cup \{\perp\} \end{array}$$

given by (whenever $\tau_j(k)$ is not defined below, we have $\tau_j(k) = \bot$):

$$
\begin{array}{lllllllllllll}
\tau_{\mathrm{I}}(1) & = & 1 & & & & & & & & & & \\
\tau_{\mathrm{II}}(1) & = & 2 & & & & & & & & & & \\
\tau_{\mathrm{III}}(1) & = & 2 & \tau_{\mathrm{III}}(2) & = & 1 & & & & & & & \\
\tau_{\mathrm{IV}}(1) & = & 2 & \tau_{\mathrm{IV}}(2) & = & 1 & & & & & & & \\
\tau_{\mathrm{V}}(1) & = & 2 & \tau_{\mathrm{V}}(2) & = & 1 & \tau_{\mathrm{V}}(3) & = & 3 & & & & \\
\tau_{\mathrm{VI}}(1) & = & 2 & \tau_{\mathrm{VI}}(2) & = & 1 & \tau_{\mathrm{VI}}(3) & = & 4 & & & & \\
\tau_{\mathrm{VII}}(1) & = & 2 & \tau_{\mathrm{VII}}(2) & = & 1 & \tau_{\mathrm{VII}}(3) & = & 4 & \tau_{\mathrm{VII}}(4) & = & 3 & \\
\tau_{\mathrm{VIII}}(2) & = & 1 & \tau_{\mathrm{VIII}}(3) & = & 4 & \tau_{\mathrm{VIII}}(4) & = & 2 & & & & \\
\tau_{\mathrm{IX}}(1) & = & 3 & \tau_{\mathrm{IX}}(2) & = & 1 & \tau_{\mathrm{IX}}(3) & = & 4 & \tau_{\mathrm{IX}}(4) & = & 2 & \\
\tau_{\mathrm{X}}(1) & = & 3 & \tau_{\mathrm{X}}(2) & = & 1 & \tau_{\mathrm{X}}(3) & = & 4 & \tau_{\mathrm{X}}(4) & = & 2 & \tau_{\mathrm{X}}(5) = 5
\end{array}
$$

Let

$$
\begin{aligned}
s_1 &= (1\mathrm{top}, ts_3^P) \\
s_2 &= (1\mathrm{top}, ts_3^P)(2\mathrm{top}, ts_3^P) \\
s_3 &= (2\mathrm{top}, ts_3^P)(\mathrm{rest}, ts_3^P)(1\mathrm{top}, ts_3^P) \\
s_4 &= s' \\
s_5 &= s
\end{aligned}
$$

where s and s' are from Example 10.17. In fact, all s_i are the extremal strings of (trimmings of) $\mathcal{T}$ from Example 10.5.

Let

$$
\begin{aligned}
q_1 &= (\varepsilon, \rho_\perp) \\
q_2 &= (s_1, s_1, \rho_\perp, 0) \\
q_3 &= (s_1, \tau_{\mathrm{I}}) \\
q_4 &= (s_2, s_2(1), \tau_{\mathrm{II}}, 0) \\
q_5 &= (s_2, s_2, \tau_{\mathrm{III}}, 0) \\
q_6 &= (s_2, \tau_{\mathrm{III}}) \\
q_7 &= (s_3, s_3(1), \tau_{\mathrm{IV}}, 0) \\
q_8 &= (s_3, s_3(1)s_3(2), \tau_{\mathrm{IV}}, 0) \\
q_9 &= (s_3, s_3, \tau_{\mathrm{IV}}, 0) \\
q_{10} &= (s_3, \tau_{\mathrm{V}}) \\
q_{11} &= (s', s'(1), \tau_{\mathrm{VI}}, 0) \\
q_{12} &= (s', s'(1)s'(2), \tau_{\mathrm{VI}}, 0) \\
q_{13} &= (s', s'(1)s'(2)s'(3), \tau_{\mathrm{VI}}, 0) \\
q_{14} &= (s', s', \tau_{\mathrm{VII}}, 0) \\
q_{15} &= (s', \tau_{\mathrm{VII}}) \\
q_{16} &= (s, s(1), \tau_{\mathrm{VIII}}, 0) \\
q_{17} &= (s, s(1)s(2), \tau_{\mathrm{VIII}}, 0) \\
q_{18} &= (s, s(1)s(2)s(3), \tau_{\mathrm{VIII}}, 0) \\
q_{19} &= (s, s(1)s(2)s(3)s(4), \tau_{\mathrm{IX}}, 0) \\
q_{20} &= (s, s, \tau_{\mathrm{IX}}, 0) \\
q_{21} &= (s, \tau_{\mathrm{X}})
\end{aligned}
$$

The sequence $q_1, \ldots, q_{21}$ induces an accepting computation, since s is completed, and therefore the state q_{21} is accepting.

10.5 $\mathscr{L}_{\mathrm{str}}(\varphi) \subseteq \mathscr{L}(\mathcal{A}^\varphi)$

With our automaton constructed, we are ready to start the proof of Theorem 10.1. We show that the string projection of every $\mathcal{D} \models \varphi$ is accepted by the automaton by showing an accepting computation based on the extremal strings induced by the $\mathcal{D}$-task word that describes which existential constraints are satisfied by the elements of $\mathcal{D}$.

Lemma 10.49. *Let $\mathcal{D}$ be a data word with* $\mathrm{string}(\mathcal{D}) = u$ *such that $\mathcal{D} \models \varphi$. Then $u \in \mathscr{L}(\mathcal{A}^{\varphi})$.*

The proof is structured as follows. Recall that the states of $\mathcal{A}^{\varphi}$ contain a pebble assignment τ. We define coherent configurations $\pi = (q, \rho, N)$ in which the pebble assignment ρ, which assigns pebbles to positions in the input, agrees with the pebble assignment τ of q. We use an inductive argument; we show that if $\mathcal{A}^{\varphi}$ runs on the projection of a data word $\mathcal{D}$ with a perfect task word, and if $\mathcal{A}^{\varphi}$ has read all the positions corresponding to data values which are not maximal while ending in a coherent configuration, then $\mathcal{A}^{\varphi}$ can read the remaining positions. We later apply this inductively on all the trimmings of a $\mathcal{D}$-task word $\mathcal{T}$ where $\mathcal{D} \models \varphi$ to construct an accepting computation of $\mathcal{A}^{\varphi}$ on the string projection of $\mathcal{D}$.

Coherent configurations Let $\mathcal{D}$ be a data word with $\mathrm{string}(\mathcal{D}) = u$. Let $\mathcal{T}$ be a perfect $\mathcal{D}$-task word, $w = \mathsf{abst}(\mathcal{T})$, and $s = \mathsf{ext}(w)$. Let ρ be an $(m+1, |u|)$-pebble assignment, τ an $(m+1, |s|)$-pebble assignment, $q = (s, \tilde{s}, \tau, b)$ or $q = (s, \tau)$ be a state and $\pi = (q, \rho, N)$ a configuration on u. π is *w-coherent* if for every $k \in [m]$ such that $\tau(k) \neq \bot$, $\rho(k) = \mathsf{Emb}_{\mathsf{ext},w}(\tau(k))$.

The next lemma essentially states that if $\mathcal{A}^{\varphi}$ has read all non-maximal positions and arrived in a coherent configuration, then there is some reachable coherent configuration with all positions read.

Lemma 10.50. *Let $\mathcal{D}$ be a data word with* $\mathrm{string}(\mathcal{D}) = u$*, let $\mathcal{T}$ be a perfect $\mathcal{D}$-task word, $w = \mathsf{abst}(\mathcal{T})$, $s = \mathsf{ext}(w)$, and let $\pi = ((s, s(1), \tau, 0), \rho, \mathsf{Pos}_{<1\mathrm{top}}(w))$ be a w-coherent configuration on u. There is a w-coherent configuration $\pi' = ((s, \tau'), \rho', [|u|])$ on u such that $\pi \rightsquigarrow_u^{\star} \pi'$.*

Proof. The proof revolves around establishing the relationship between the positions of w, which originates from the input, and the positions of s, which is encoded in $\mathcal{A}^{\varphi}$. Let $C \in \mathbb{N}$ and $1 \leq \ell_1 < \cdots < \ell_C \leq |u|$ such that

$$\{\ell_1, \ldots, \ell_C\} = \mathsf{ExtPos}(w) \cup \mathsf{Pos}_{1\mathrm{top}}(w).$$

That is, C is the number of positions which are extremal or have maximal data value. For every $c \in \{0\} \cup [C]$, let $N_c = \mathsf{Pos}_{<1\mathrm{top}} \cup \{\ell_1, \ldots, \ell_c\}$. We have $N_0 = \mathsf{Pos}_{<1\mathrm{top}}$ and $N_{c+1} = N_c \cup \{\ell_c\}$ for $c \neq 0$. Note that we may have $\ell_c \in N_c$ when ℓ_c is extremal. Let $a_0 = 0$. For every $c \in \{0\} \cup [C-1]$, let:

$$a_{c+1} = \begin{cases} a_c + 1, & \ell_{c+1} \in \mathsf{ExtPos}(w) \\ a_c, & \ell_{c+1} \notin \mathsf{ExtPos}(w) \end{cases}$$

That is, a_c counts how many of $\{\ell_1, \ldots, \ell_c\}$ are extremal positions. Observe that ℓ_C is necessarily in $\mathsf{ExtPos}(w)$, since by definition it is the rightmost position with maximal data value. Similarly, $\ell_1 \in \mathsf{ExtPos}(w)$. Consequently, $s(1) \cdots s(a_C) = s$.

We give a construction of a computation $(\bar{t}, \bar{\pi})$ with transitions $\bar{t} = (t_1, \ldots, t_C)$ and w-coherent configurations $\bar{\pi} = (\pi_0, \ldots, \pi_C)$ on u such that $\pi_0 = \pi$, and $\pi_c \rightsquigarrow_u^{t_{c+1}} \pi_{c+1}$ for all $c \in \{0\} \cup [C-1]$. We show this by describing how every position $\{\ell_1, \ldots, \ell_C\}$ was either correctly consumed in a previous step, or is correctly consumed in the current step. Let $\pi_c = (q_c, \rho_c, N_c)$ where $q_c = (s, s(1) \cdots s(a_c + 1), \tau_c, b_c)$ for $c < C$ and $q_C = (s, \tau_C)$. We construct t_{c+1} and π_{c+1} inductively for $c \in \{0\} \cup [C-1]$ by dividing into cases as follows.

- Assume $\ell_{c+1} \in \mathsf{ExtPos}(w)$. We have $\ell_{c+1} = \mathsf{Emb}_{\mathsf{ext},w}(a_{c+1})$ and hence $w(\ell_{c+1}) = s(a_{c+1})$. If $c+1 < C$, let $b_{c+1} = 0$.

  1. Assume $s(a_{c+1}) \notin \Gamma_{1\mathrm{top}}$. Then we assume it was correctly consumed in a previous step, and let $t_{c+1} = (q_c, q_{c+1})$, $\rho_{c+1} = \rho_c$, and $\tau_{c+1} = \tau_c$. Then $t_{c+1} \in \delta$, $\pi_c \rightsquigarrow_u^{t_{c+1}} \pi_{c+1}$, and π_{c+1} is a w-coherent configuration on u.
  2. Assume $s(a_{c+1}) \in \Gamma_{1\mathrm{top}}$. By Observation 10.48, there is some available pebble $k \in [m]$ in q_c. We will use it to read the base letter of $s(a_{c+1})$, which in our notation is given by $\xi^{\omega(s(a_{c+1}))}$ (recall that we are reading positions of the input u, whereas the letters of s are annotated with tasks and whether they are completed or promised). Let $\tau_{c+1} = \tau_c[k \mapsto a_{c+1}]$. There exist $i, j \in [m] \cup \{\triangleright, \triangleleft\}$ and
  $$t_{c+1} = (q_c, k\text{-MOVE}_{i,j}, \xi^{\omega(s(a_{c+1}))}, q_{c+1})$$
  such that $t_{c+1} \in \delta$. By the choice of i and j in the definition of an extremal transition from a prefix state in Section 10.4, $\hat{\tau}_c(i) < a_{c+1} \leq \hat{\tau}_c(j)$. Since π_c is w-coherent and $\mathsf{Emb}_{\mathsf{ext},w}$ is order-preserving, $\hat{\rho}_c(i) < \ell_{c+1} \leq \hat{\rho}_c(j)$. We have $\ell_{c+1} \notin N_c$, hence no pebble has been placed on ℓ_{c+1}. In particular we have $\ell_{c+1} \neq \rho_c(j)$, implying that $\hat{\rho}_c(i) < \ell_{c+1} < \hat{\rho}_c(j)$. Let $\rho_{c+1} = \rho_c[k \mapsto a_{c+1}]$. Since $s(a_{c+1}) = w(\ell_{c+1})$, $\xi^{\omega(s(a_{c+1}))} = \xi^{\omega(\ell_{c+1})} = u(\ell_{c+1})$. Then $\pi_c \rightsquigarrow_u^{t_{c+1}} \pi_{c+1}$ and π_{c+1} is w-coherent.

- Now assume $\ell_{c+1} \notin \mathsf{ExtPos}(w)$. Then $\ell_{c+1} \in \mathsf{Pos}_{1\mathrm{top}}(w)$, $\ell_{c+1} \notin N_c$, $a_{c+1} = a_c$, and for every $\theta \in \Theta_\exists$ there are $1 \leq c_{\theta,l} < c < c_{\theta,r} \leq C$ such that
  $$\begin{aligned} c_{\theta,l} &= \max(\{\tilde{c} \in [C] \mid \ell_{\tilde{c}} \in \mathsf{ExtPos}_{1\mathrm{top},\theta}(w))\} \cap [c-1]), \\ c_{\theta,r} &= \min(\{\tilde{c} \in [C] \mid \ell_{\tilde{c}} \in \mathsf{ExtPos}_{1\mathrm{top},\theta}(w))\} \cap \{c+1, \ldots, C\}). \end{aligned}$$
  Let s_{pr} and s_{su} be respectively the prefix and suffix of s given by $s_{pr} = s(1) \cdots s(a_{c+1})$ and $s_{su} = s(a_{c+1}+1) \cdots s(|s|)$. We have $s = s_{pr} s_{su}$ and $\mathsf{ext}(s) = \mathsf{ext}(s_{pr} w(\ell_{c+1}) s_{su})$, and hence $w(\ell_{c+1}) \in \Gamma_{1\mathrm{top}}^{\mathsf{not\,ext}}(s, a_c + 1)$. Let $\tau_{c+1} = \tau_c$ and $b_{c+1} = 1$. There exists i such that
  $$(q_c, (m+1)\text{-MOVE}_{i,\triangleleft}, \xi^{\omega(w(\ell_{c+1}))}, q_{c+1}) \in \delta.$$
  If $b_c = 0$, then by the choice of i in the definition of a non-extremal transition from a prefix state in Section 10.4, $\hat{\tau}_c(i) \leq a_c$. Since $\mathsf{Emb}_{\mathsf{ext},w}$ is order-preserving, $\hat{\rho}_c(i) \leq \ell_c$

and hence $\hat{\rho}_c(i) < \ell_{c+1}$. If $b_c = 1$, then the computation $((t_1, \ldots, t_c), (\pi_0, \ldots, \pi_c))$ has at least one non-extremal transition. Hence the pebble $m+1$ was moved during this computation, i.e. $\rho_c(m+1) \in \{\ell_1, \ldots, \ell_c\}$. In either case, $\hat{\rho}_c(i) < \ell_{c+1} < \hat{\rho}_c(\triangleleft) = |u| + 1$. Let $\rho_{c+1} = \rho_c$, and note that $\xi^{\omega(\ell_{c+1})} = u(\ell_{c+1})$. Then $\pi_c \rightsquigarrow_u^{t_{c+1}} \pi_{c+1}$, and since $\tau_{c+1} = \tau_c$ and ρ_{c+1} agrees with ρ_c on all $k \in [m]$, we have that π_{c+1} is w-coherent.

The lemma follows with $t' = t_C$ and $\pi' = \pi_C$. □

Induced sequences of extremal strings Let $\mathcal{D}$ be a data word with $h = maxval_{\mathcal{D}}$, and $\mathcal{T}$ a $\mathcal{D}$-task word. For every $e \in \{0\} \cup [h]$, let $s_e = \mathsf{ext}(\mathsf{abst}(\mathcal{T}^{\setminus\setminus h-e}))$. We say the sequence $s_0, \ldots, s_h$ is *induced by* $\mathcal{T}$.

Proof of Lemma 10.49 Let $R = maxval_{\mathcal{D}}$. Since $\mathcal{D} \models \varphi$, by Proposition 10.32, there exists a perfect completed $\mathcal{D}$-task word $\mathcal{T}$. Notice that the sequence $s_0, \ldots, s_R$ of extremal strings induced by $\mathcal{T}$ satisfies that $s_0 = \varepsilon$, s_R is completed, and (s_{r-1}, s_r) is consecutive for $r \in [R]$, by definition. For every $r \in \{0\} \cup [R]$, let u_r be the string projection of $\mathcal{D}^{\setminus R-r}$ and let $w_r = \mathsf{abst}(\mathcal{T}^{\setminus\setminus R-r})$. We have $s_r = \mathsf{ext}(w_r)$. Since $\mathcal{T}$ is perfect, so are its iterated trimming $\mathcal{T}^{\setminus\setminus R-r}$ and s_r.

Let $\pi_0 = \pi_{\mathrm{init}}$. Observe that for every $r \in \{0\} \cup [R]$, π_0 is a w_r-coherent configuration on u_r. We construct a sequence of transitions $(t_1, \ldots, t_R)$ and a sequence of configurations $(\pi_1, \ldots, \pi_R)$ such that, for every $r \in [R]$, $\pi_r = ((s_r, \tau_r), \rho_r, [|r|])$ is a w_r-coherent configuration on u_r and $\pi_0 \rightsquigarrow_{u_r}^{\star} \pi_r$. We construct the transitions and configurations inductively as follows. For every $r \in [R]$, assume there are t_r and π_r as described above.

The universe of $\mathcal{T}^{\setminus\setminus R-r}$ is a subset of the universe of $\mathcal{T}^{\setminus\setminus R-(r+1)}$. We write $\mathsf{Emb}_{w_r \hookrightarrow w_{r+1}}$ for the embedding obtained by the composition of $\mathsf{Emb}_{\mathsf{abst}, \mathcal{T}^{\setminus\setminus R-r}}$ and $\mathsf{Emb}^{-1}_{\mathsf{abst}, \mathcal{T}^{\setminus\setminus R-(r+1)}}$, that is, the embedding of positions of w_r to positions in w_{r+1}. The string u_r is a substring of u_{r+1} since $\mathcal{T}^{\setminus\setminus R-(r)}$ is a substructure of $\mathcal{T}^{\setminus\setminus R-(r+1)}$. We denote by $\mathsf{pos}_{r,r+1}$ the mapping of positions of u_r to positions of u_{r+1}. Since the universe of $\mathcal{T}^{\setminus\setminus R-e}$ is equal to the universe of $\mathcal{D}^{\setminus R-e}$ for $e \in \{r, r+1\}$, $\mathsf{Emb}_{w_r \hookrightarrow w_{r+1}} = \mathsf{pos}_{r,r+1}$.

Then we can talk about the configuration where the pebble assignment is updated using this mapping: let $\tilde{\pi}_r = ((s_r, \tau_r), \tilde{\rho}_r, \mathsf{Pos}_{<1\mathrm{top}}(w_{r+1}))$ be a configuration on u_{r+1} with

$$\tilde{\rho}_r(k) = \begin{cases} \bot, & \rho_r(k) = \bot \\ \mathsf{Emb}_{w_r \hookrightarrow w_{r+1}}(\rho_r(k)), & \rho_r(k) \neq \bot \end{cases}$$

The semantics of PIA allow us to lift a computation from a substring to a string, thus $\pi_0 \rightsquigarrow_{u_{r+1}}^{\star} \tilde{\pi}_r$.

Let π_{r+1}^{pr} be the configuration on u_{r+1} given by

$$\pi_{r+1}^{pr} \quad = \quad ((s_{r+1}, s_{r+1}(1), \tau_{r+1}^{pr}, 0), \tilde{\rho}_r, \mathsf{Pos}_{<1\mathrm{top}}(w_{r+1}))$$

where for every pebble $k \in [m]$, $\tau^{pr}_{r+1}(k)$ is $\perp$ if $\tau_r(k)$ is not in the image of $\mathsf{PEmb}_{s_{r+1} \hookrightarrow s_r}$, and is $(\mathsf{PEmb}_{s_{r+1} \hookrightarrow s_r})^{-1}(\tau_r(k))$ otherwise. Let $t_{r+1} = ((s_r, \tau_r), (s_{r+1}, s_{r+1}(1), \tau^{pr}_{r+1}, 0))$. Since the pair (s_r, s_{r+1}) is consecutive, we get that $t_{r+1} \in \delta$ and $\tilde{\pi_r} \rightsquigarrow^{t_{r+1}}_{u_{r+1}} \pi^{pr}_{r+1}$, implying that $\pi_0 \rightsquigarrow^{\star}_{u_{r+1}} \pi^{pr}_{r+1}$. We prove that $\pi^{pr}_{r+1}(k)$ is w_{r+1}-coherent. π_r is w_r-coherent by the assumption, hence by definition $\rho_r(k) = \mathsf{Emb}_{\mathsf{ext},w_r}(\tau_r(k))$.

Let $k \in [m]$ be such that $\tau^{pr}_{r+1}(k) \neq \perp$. Then $\tau_r(k)$ is in the image of $\mathsf{PEmb}_{s_{r+1} \hookrightarrow s_r}$ and in particular $\tau_r(k) \neq \perp$, and $\tau^{pr}_{r+1}(k)$ is given by:

$$\mathsf{Emb}^{-1}_{\mathsf{ext} \circ \mathsf{abst}, \mathcal{T}^{\setminus R-(r+1)}} \left(\mathsf{Emb}_{\mathsf{ext} \circ \mathsf{abst}, \mathcal{T}^{\setminus R-r}}(\tau_r(k)) \right) =$$
$$\mathsf{Emb}^{-1}_{\mathsf{ext}, w_{r+1}} \left(\mathsf{Emb}_{w_r \hookrightarrow w_{r+1}} (\rho_r(k)) \right).$$

We have

$$\mathsf{Emb}_{\mathsf{ext},w_{r+1}}(\tau^{pr}_{r+1}(k)) = \mathsf{Emb}_{w_r \hookrightarrow w_{r+1}} (\rho_r(k)) = \tilde{\rho}_r(k)$$

Hence, $\pi^{pr}_{r+1}(k)$ is w_{r+1}-coherent.

Now we apply Lemma 10.50 with the data word $\mathcal{D}^{\setminus R-(r+1)}$ of size n_{r+1}, the string projection u_{r+1}, the perfect $\mathcal{D}^{\setminus R-(r+1)}$-task word $\mathcal{T}^{\setminus R-(r+1)}$, the abstraction w_{r+1}, the extremal string s_{r+1}, and the configuration $\pi^{pr}_{r+1}(k)$; we get that there are τ_{r+1} and ρ_{r+1} such that π_{r+1} is a w_{r+1}-coherent configuration on u_{r+1} and $\pi^{pr}_{r+1}(k) \rightsquigarrow^{\star}_{u_{r+1}} \pi_{r+1}$, and therefore $\pi_0 \rightsquigarrow^{\star}_{u_{r+1}} \pi_{r+1}$. The lemma follows for the computation $\pi_0 \rightsquigarrow^{\star}_{u_R} \pi_R$ since $u = u_{r+1}$, $n_R = |u|$, and $(s_R, \tau_R) \in F$.

10.6 $\mathscr{L}(\mathcal{A}^{\varphi}) \subseteq \mathscr{L}_{\mathrm{str}}(\varphi)$

We now prove the second direction of Theorem 10.1.

Lemma 10.51. *Let $w \in \Xi^{\star}$ be accepted by $\mathcal{A}^{\varphi}$. Then $w \in \mathscr{L}_{\mathrm{str}}(\varphi)$.*

Proof. Given $w \in \mathscr{L}(\mathcal{A}^{\varphi})$, we build a data word $\mathcal{D}$ for it based on an accepting computation of $\mathcal{A}^{\varphi}$ on w. To show that $\mathcal{D} \models \varphi$, we prove the existence of a perfect completed $\mathcal{D}$-task word based on the syntactic representation of consecutive extremal strings.

Let $(\bar{t}, \bar{\pi})$ be an accepting computation. Let z be the number of transitions from a state in $\{q_{\mathrm{init}}\} \cup Q_e$ to a state in Q_p in $\bar{t}$. This will be the number of different data values in the constructed data word. The computation can be broken down into parts as follows:

$$\pi_0 \rightsquigarrow^{t_0}_{w} \pi^{(1)}_1 \rightsquigarrow^{\bar{t}_1}_{w} \pi^{(2)}_1 \rightsquigarrow^{\bar{t}_2}_{w} \cdots \rightsquigarrow^{\bar{t}_{h-1}}_{w} \pi^{(z)}_1 \rightsquigarrow^{\bar{t}_z}_{w} \pi_f$$

where the target state of any transition is in Q_e if and only if there is $e \in [z]$ such that the transition is the last one in $\bar{t}_e$. The sequence $\bar{t}$ is equal to the concatenation of t_0 and the sequences $\bar{t}_1, \ldots, \bar{t}_z$. For every e and a, let the state of $\pi^{(e)}_a$ be $(s_e, s_e(1) \cdots s_e(a_e), \tau_{e,a_e}, b_e)$.

For every transition $t = t^{(e)}_a$, let $\gamma_t \in \Gamma_{1\mathrm{top}} \cup \{\varepsilon\}$ be:

1. if $t \in Q \times Q$, $\gamma_t = \varepsilon$.
2. Otherwise, if t is a non-extremal transition, let $\gamma_t \in \Gamma_{1\mathrm{top}}^{\mathsf{not\,ext}}(s_e, a_e)$ such that $t \in Q \times \textsc{move}_{m+1} \times \{\xi^{\gamma_t}\} \times Q$. If t is an extremal transition, let $\gamma_t = s(a_e)$ (and note that we again have that $t \in Q \times \textsc{move}_{m+1} \times \{\xi^{\gamma_t}\} \times Q$).

Let $\gamma_{\bar{t}_e} = \gamma_{t_1^{(e)}} \cdots \gamma_{t_{len(e)}^{(e)}}$, where $len(e) = |\bar{t}_e|$. For every $e \in [z]$, let v_e, u_e be the substrings of w which the automaton reads during the transitions $\bar{t}_e$ respectively $(t_0, \bar{t}_1, \ldots, \bar{t}_e)$. Let g'_e be such that u_e is the shuffle $v_e \sqcup\!\sqcup_{g'_e} u_{e-1}$ relative to the positions in w.

We prove the following by induction on e (the proof is given in the next section): there is a data word $\mathcal{D}_e$ and a $\mathcal{D}_e$-task word $\mathcal{T}_e$ such that for every $p \leq e$ we have:

(i) $\mathrm{string}(\mathcal{D}_e{}^{\backslash p}) = u_{e-p}$,

(ii) $\mathsf{ext}(\mathsf{abst}(\mathcal{T}_e{}^{\backslash\backslash p})) = s_{e-p}$,

(iii) $\mathcal{T}_e{}^{\backslash\backslash p} = \mathcal{T}_{e-p}$ and $\mathcal{T}_e{}^{\backslash\backslash p}$ is a $\mathcal{D}_e{}^{\backslash p}$-task word,

(iv) the universe of $\mathcal{D}_e$ is contained in $[e] \times \mathbb{N}$.

Since for every $e \in [z]$, s_e is an extremal string appearing in a state in Q_p, s_e is perfect. Therefore $\mathcal{T}_z$ is a perfect $\mathcal{D}$-task word. Since the computation is accepting, the extremal string s_z is complete and therefore $\mathcal{T}_z$ is a completed $\mathcal{D}$-task word. By Prop. 10.32, we have $\mathcal{D} \models \varphi$. □

10.6.1 Inductive proof in Lemma 10.51

We assume the induction hypothesis for $e-1$ and prove for e. Let $r_e \in \Gamma^\star_{1\mathrm{top}} \cap \Gamma^\star_{\mathrm{P}}$ be obtained from $\gamma_{\bar{t}_e}$ by setting all tasks to P. Notice $v_e = \xi^{\gamma_{\bar{t}_e}} = \xi^{\gamma_{r_e}}$. Let $g_e : [|r_e|] \to [|r_e| + |s_{e-1}|]$ be given by $g_e(\ell) = \ell + |\mathsf{ExtPos}(\mathsf{abst}(\mathcal{T}_{e-1})) \cap [g'_e(\ell) - \ell]|$. Recall that $\bar{g}_e : [|s_{e-1}|] \to [|r_e| + |s_{e-1}|]$. Note:

$$\begin{aligned} &\forall \ell \in [|r_e|], \quad \xi^{(r_e \downarrow_{g_e} s_{e-1})(g_e(\ell))} = (v_e \downarrow_{g'_e} u_{e-1})(g'_e(\ell)) \\ &\forall \ell \in [|s_{e-1}|], \quad \xi^{(r_e \downarrow_{g_e} s_{e-1})(\bar{g}_e(\ell))} = (v_e \downarrow_{g'_e} u_{e-1})(\bar{g}'_e(\ell)) \end{aligned}$$

Claim 10.52. *There is a data word $\mathcal{D}_e$ such that* $\mathrm{string}(\mathcal{D}_e) = u_e$ *and the universe of $\mathcal{D}_e$ is contained in $[e] \times \mathbb{N}$, and there exists a $\mathcal{D}_e$-task word $\mathcal{T}_e$ such that* $\mathsf{ext}(\mathsf{abst}(\mathcal{T}_e)) = \mathsf{ext}(r_e \downarrow_{g_e} s_{e-1})$ *and for every $p \leq e$, $\mathcal{T}_e{}^{\backslash\backslash p} = \mathcal{T}_{e-p}$.*

Proof. Let D_{e-1} be the universe of $\mathcal{D}_{e-1}$. Let $\mathcal{D}_e$ be the data word over Ξ with universe $D_{e-1} \cup (\{e\} \times [|r_e|])$ such that:

1. $\mathcal{D}_{e-1}$ is the substructure of $\mathcal{D}_e$ induced by D_{e-1}.
2. For every $\ell \in [|r_e|]$ and $r_e(\ell) = (h, ts)$, $\mathcal{D}_e \models \xi^{\omega(ts)}(e, \ell)$.

3. For every $\ell_1, \ell_2 \in [|r_e|]$, $\mathcal{D}_e \models (e, \ell_1) \sim_2 (e, \ell_2)$.

4. For every $d \in D_{e-1}$ and $\ell \in [|r_e|]$, $\mathcal{D}_e \models d <_2 (e, \ell)$.

5. For every $\ell_1, \ell_2 \in [|r_e|]$, $\mathcal{D}_e \models (e, \ell_1) \leq_1 (e, \ell_2)$ if and only if $\ell_1 \leq \ell_2$.

6. For every $d \in D_{e-1}$ and $\ell \in [|r_e|]$, $\mathcal{D} \models d \leq_1 (e, \ell)$ if and only if $g'_e(\ell) \geq |\{d' \in D_{e-1} \mid \mathcal{D}_{e-1} \models d' \leq_1 d\}| + \ell$.

Let $\mathcal{T}_e$ be the $\mathcal{D}_e$-task word such that

- for every $d \in D_{e-1}$, there are $ts, ts' \in 2^{\mathrm{Tasks}}_{\Omega}$ such that $\omega(ts) = \omega(ts')$, $\mathcal{T}_e \models ts(d)$, and $\mathcal{T}_{e-1} \models ts'(d)$, and
- for every $(e, \ell) \in D_e$, there are $ts, ts' \in 2^{\mathrm{Tasks}}_{\Omega}$ such that $\omega(ts) = \omega(ts')$, $r_e(\ell) = (1\mathrm{top}, ts')$, and $\mathcal{T}_e \models ts(e, \ell)$.

By our construction, $\mathcal{T}_e{}^{\setminus\setminus p} = \mathcal{T}_{e-p}$ for $p = 1$. For $p > 1$, this equality follows from the induction hypothesis.

Clearly $\mathsf{ext}(\mathsf{abst}(\mathcal{T}_e{}^{\setminus\setminus 1})) = \mathsf{ext}(\mathsf{abst}(\mathcal{T}_{e-1})) = s_{e-1}$, by the induction hypothesis. We apply Lemma 10.45 with $\mathcal{D}_e$ for $\mathcal{D}$, $\mathcal{T}_e$ for $\mathcal{T}$ and r_e for r. Note that g in the lemma is g_e, hence we get $\mathsf{ext}(\mathsf{abst}(\mathcal{T}_e)) = \mathsf{ext}(r_e{\downarrow_{g_e}} s_{e-1})$.

Let $\mathcal{D}_{\max}$ be the substructure of $\mathcal{D}_e$ which consists of the elements of $\mathcal{D}_e$ with maximal data value. By the definition of $\mathcal{D}_e$, we have $\mathrm{string}(\mathcal{D}_{\max}) = \xi^{r_e} = v_e$. By induction, $\mathrm{string}(\mathcal{D}_{e-1}) = u_{e-1}$. By the definition g'_e, $u_e = v_e \sqcup_{g'_e} u_{e-1}$, and hence $\mathrm{string}(\mathcal{D}_e) = u_e$.

□

From Lemma 10.35, it follows that there exists $f_e : [|r_e|] \to [|r_e| + |s_{e-1}|]$ such that s_e is a substring of $r_e{\downarrow_{f_e}} s_{e-1}$, hence $s_e = \mathsf{ext}(r_e{\downarrow_{f_e}} s_{e-1})$.

Let $r'_e = \mathsf{ext}(r_e)$ and $s'_{e-1} = \mathsf{ext}(\downarrow s_{e-1})$. There exist $G_e, F_e : [|r'_e|] \to [|r'_e| + |s'_{e-1}|]$ such that $\mathsf{ext}(r_e{\downarrow_{f_e}} s_{e-1}) = r'_e \sqcup_{F_e} s'_{e-1}$ and $\mathsf{ext}(r_e{\downarrow_{g_e}} s_{e-1}) = r'_e \sqcup_{G_e} s'_{e-1}$. Assume that $F_e \neq G_e$. Recall that $\bar{F}_e, \bar{G}_e : [|s'_{e-1}|] \to [|r'_e| + |s'_{e-1}|]$. Let $\tilde{\ell} \in [|r'_e| + |s'_{e-1}|]$ be the length of the maximal common prefix of $r'_e \sqcup_{F_e} s'_{e-1}$ and $r'_e \sqcup_{G_e} s'_{e-1}$. We divide into two cases:

- Assume the letter at position $\tilde{\ell} + 1$ in $r'_e \sqcup_{G_e} s'_{e-1}$ is in $\Gamma_{1\mathrm{top}}$. Then the letter at position $\tilde{\ell} + 1$ in $r'_e \sqcup_{F_e} s'_{e-1}$ is not in $\Gamma_{1\mathrm{top}}$. Let $\ell_1 \in [|r'_e|]$ and $\ell_2 \in [|s'_{e-1}|]$ be such that $G_e(\ell_1) = \bar{F}_e(\ell_2) = \tilde{\ell} + 1$. Then $F_e(\ell_1) > \bar{F}_e(\ell_2)$ and $\bar{G}_e(\ell_2) > G_e(\ell_1)$.
- Assume the letter at position $\tilde{\ell} + 1$ in $r'_e \sqcup_{F_e} s'_{e-1}$ is in $\Gamma_{1\mathrm{top}}$. Analogously to the previous case, we have $G_e(\ell_1) > \bar{G}_e(\ell_2)$ and $\bar{F}_e(\ell_2) > F_e(\ell_1)$.

In both cases, $G_e(\ell_1) < \bar{G}_e(\ell_2)$ if and only if $\bar{F}_e(\ell_2) < F_e(\ell_1)$.

Let $t_c^{(e)}$ be the transition in which the automaton reads position $\mathsf{Emb}_{\mathsf{ext},r_e}(\ell_1)$ of v_e. Let $\ell_1^w \in [|w|]$ be the position of w which $t_c^{(e)}$ reads. Let k-MOVE$_{i,j}$ be the move in $t_c^{(e)}$. There is a pebble $k' = \tau_{e,c}^{-1}(\bar{F}_e(\ell_2))$ on the position in s_e corresponding to ℓ_2 in s'_{e-1}. Let $\ell_2^w \in [|w|]$ be the position of k' in w. G_e and $\bar{G}_e$ have disjoint images, hence $G_e(\ell_1) \neq \bar{G}_e(\ell_2)$, and similarly for $F_e, \bar{F}_e$. We divide into cases:

1. If $G_e(\ell_1) < \bar{G}_e(\ell_2)$, then $\bar{F}_e(\ell_2) < F_e(\ell_1)$. By the definition of the automaton, the pebble i is located to the left of ℓ_1^w, and i is either k' itself, or another pebble located to the right of k'. Hence, $\ell_1^w > \ell_2^w$. But since $G_e(\ell_1) < \bar{G}_e(\ell_2)$, we have $g_e(\mathsf{Emb}_{\mathsf{ext},r_e}(\ell_1)) < \bar{g}_e(\mathsf{Emb}_{\mathsf{ext},\downarrow s_{e-1}}\ell_2)$, we have $g'_e(\mathsf{Emb}_{\mathsf{ext},r_e}(\ell_1)) < \bar{g}'_e(\mathsf{Emb}_{\mathsf{ext},\downarrow s_{e-1}}(\ell_2))$, and hence $\ell_1^w < \ell_2^w$, in contradiction.
2. The case of $g_e(\ell_1) > \bar{g}_e(\ell_2)$ is analogous.

Hence $F_e = G_e$ and $s_e = \mathsf{ext}(r_e{\downarrow_{g_e}} s_{e-1})$, and the induction hypothesis holds.

10.7 Complexity discussion

The previous two sections showed that $\mathscr{L}_{\mathrm{str}}(\varphi) = \mathscr{L}(\mathcal{A}^\varphi)$. In this section, we show that our construction provides an automata-theoretic proof to $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ having an ExpSpace satisfiability problem. First, we assert the size of $\mathcal{A}^\varphi$.

Lemma 10.53. *Let $n = |\psi|$ be the size of the original (not necessarily in normal form) formula ψ. The size of $\mathcal{A}^\varphi$ is at most double exponential in n, while the number of pebbles m is at most exponential in n.*

Proof. By Theorem 9.13, $|\Xi|$ and $|\Theta|$ and hence m are exponential in n. The size of Γ is therefore double exponential in n. By Lemma 10.19, $\mathsf{EXT}(\Gamma) \subseteq \Gamma^{7\cdot|\Theta_\exists|}$, which is double exponential in n. Given an extremal string s, the number of $(m+1, |s|)$-pebble assignments is at most $(|s|+1)^{m+1}$, which is double exponential in n. Hence $|Q|$ is double exponential. Since $\delta \subseteq (Q \times Q) \cup (Q \times \text{MOVE}_{m+1} \times \Sigma \times Q)$, $|\delta|$ is double exponential. □

Next, we describe how to search for an accepting computation in $\mathcal{A}^\varphi$, which amounts to searching for a sequence of consecutive extremal strings ending in a completed extremal string. We build this sequence starting from the initial extremal string by nondeterministically choosing the next extremal string based on the syntactic representation of consecutive extremal strings. Given s_e and s_{e+1}, checking if (s_e, s_{e+1}) are consecutive in ExpSpace is done as follows. We iterate over all $r \in \Gamma^\star_{1\mathrm{top}} \cap \Gamma^\star_{\mathrm{C}}$ such that $|r| \leq 7|\Theta_\exists|$ and over all strictly monotone functions $g : [|r|] \to [|r| + |s_e|]$. We search for such r and g for which $s_{e+1} = r{\downarrow_g} s_e$ and answer to whether such r and g are found. Lemmas 10.45 and 10.46 guarantee the correctness of a semi-decision procedure behaving as above

without restricting the length of r. To guarantee we non-deterministically explore the entire transition relation, we use the following lemma, which allows us to bound the search space by limiting our attention to r which expand the current string with extremal positions:

Lemma 10.54. *Let $\mathcal{T}$ be a $\mathcal{D}$-task word. Let d be an element of $\mathcal{T}$ not in $\mathsf{ExtElem}(\mathcal{T})$ such that $value_{\mathcal{D}}(d) = maxval_{\mathcal{D}}$. Let $\mathcal{D}_{-d}$ and $\mathcal{T}_{-d}$ be the substructures of $\mathcal{D}$ respectively $\mathcal{T}$ obtained by removing d. Then $\mathcal{T}_{-d}$ is a $\mathcal{D}_{-d}$-task word and $\mathsf{ext}(\mathsf{abst}(\mathcal{T})) = \mathsf{ext}(\mathsf{abst}(\mathcal{T}_{-d}))$.*

Proof. Assume that $\mathcal{T}_{-d}$ is not a task word. There are $d' \in D$, $ts \in 2^{\mathrm{Tasks}}_{\Omega}$, and $\theta \in \omega(ts)$ such that $d \neq d'$, $\mathcal{T} \models ts(d)$, and $C_\theta \in ts$, but for every element $d'' \neq d$ of $\mathcal{D}_{-d}$, $\mathcal{D}_{-d} \not\models \theta(d', d'')$. However, the non-existence of such d'' is in contradiction to Lemma 10.44. Since $d \notin \mathsf{ExtElem}(\mathcal{T})$, $\mathsf{ExtElem}(\mathcal{T}) = \mathsf{ExtElem}(\mathcal{T}_{-d})$. The string $\mathsf{abst}(\mathcal{T}_{-d})$ is the substring of $\mathsf{abst}(\mathcal{T})$ obtained by deleting the letter corresponding to d, and hence $\mathsf{ext}(\mathsf{abst}(\mathcal{T})) = \mathsf{ext}(\mathsf{abst}(\mathcal{T}_{-d}))$. □

Lemma 10.55. *δ is ExpSpace$(\log(|\mathcal{A}|))$-computable.*

Proof. Let $n = |\psi|$. Using Lemma 10.53, the sizes of the representation of a state $q \in Q$, a string s in $\Gamma^{7\cdot|\Theta_\exists|}$, a pebble $k \in [m+1]$, and a letter $\gamma \in \Gamma_{1\mathrm{top}}$ are all at most exponential in n.

We can verify that s is an extremal string in exponential space by computing $\mathsf{ExtPos}(s)$ and verifying that $|\mathsf{ExtPos}(s)| = |s|$. The set $\mathsf{ExtPos}(s)$ can be computed by going over the string s and keeping track of the relevant minimum and maximum positions (of which there are an exponential number). Similarly, for $\tilde{\ell} \in [|s|]$, one can verify that $\gamma \in \Gamma^{\mathsf{not\,ext}}_{1\mathrm{top}}(s, \tilde{\ell})$.

Verifying that an extremal string s is perfect is as follows: for every two positions $\ell_1 < \ell_2$ of s such that $\{\ell_1, \ell_2\} \cap \mathsf{Pos}_{1\mathrm{top}}(s) \neq \emptyset$, it is straight-forward to compute $\mathrm{perfect}_{s(\ell_1),s(\ell_2)}(x, y)$ in exponential space. Since $\mathrm{perfect}_{s(\ell_1),s(\ell_2)}(x, y)$ is a conjunction of atoms and negations of atoms, it is also easy to complete it to its equivalent 2-type $\beta(x, y)$. We can then check that $\beta(x, y), \beta(y, x) \in \Theta_\forall$.

We saw above that is it possible to non-deterministically compute the extremal strings which are consecutive to s, and hence the set of transitions leaving $q \in Q$, in ExpSpace. □

Thus, putting this together with Theorem 10.1 and Observation 8.16 we have:

Corollary 10.56. *Finite satisfiability of $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ is in ExpSpace.*

Discussion This chapter brought together Chapters 8 and 9 by constructing a PIA for an $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ formula ψ. We had to bridge the gap between there being an unbounded number of different data values in the models of ψ and PIAs having finite memory. For this, we had our PIAs run on their input strings in rounds, with the idea

being that positions read in the same round were assumed to have the same data value. Furthermore, positions read in later rounds were assumed to have larger data value. That way, we could abstract away the precise data values originally assigned to the elements, and only consider the relationships between the data values: equality, smaller by 1, or smaller by more than 1. Since each round can have an unbounded number of positions read, we still needed to distill what was necessary for satisfaction of the formula into a bounded form. This was achieved with extremal strings, which were essentially the minimal substring the automaton must read during a round. Finally, those two ideas were embodied in our construction of a PIA from a formula given in normal form.

CHAPTER 11

Conclusion and Discussion

The main aim of this thesis was to tackle questions that arise when incorporating data values into decidable logics using automata-theoretic methods. Incorporating data values into such logics usually results in a nearly-undecidable formalism, so it is especially challenging to manage the computational complexity of reasoning tasks. Furthermore, the declarative nature of logical formulas offers little insight on how to approach such problems. In contrast, automata approaches are more behavioral, and provide intuition on how to process structures with data values.

In the first part, we considered a recently introduced description logic which allows one to specify constraints on multiple integer values associated with logical elements along a path in the model. This logic, $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, has several desirable traits which are not commonly found together in the literature. Namely, the fact that is enjoys a decidable concept satisfiability problem even in the presence of general TBoxes, while supporting access to the concrete domain via long paths of arbitrary roles, and despite the concrete domain having the (non-dense) integers as the underlying numeric set. The presence of any one of these alone usually has some consequences like higher complexity or even undecidability. The question we solved for this logic was the complexity of reasoning, as the proof of its decidability result [38] does not provide upper bounds for the problem.

We followed the approach in the decidability proof and split the task into a usual $\mathcal{ALCF}$ satisfiability check and an embeddability check of a constraint graph. The most natural way to go about performing the latter task with automata would have been to articulate a characterizing condition for embeddability, such that the condition is verifiable with an automata model that has a decidable emptiness test. Then, the automata model would accept exactly the (tree) models of the formula.

However, we could not find an automata-verifiable condition which was both necessary and sufficient for the embeddability of constraint graphs in general, as we would always find some counterexample for the proposed condition being sufficient. Notably, these

counterexamples would always be irregular in nature. Therefore, instead of going the usual route of finding a characterizing embeddability condition for *all* graphs, we used the same clever trick employed by Demri and D'Souza in [69] and relied on Rabin's Theorem. Rabin's Theorem guarantees that any Rabin-recognizable set of trees contains a regular tree, so we expressed a condition that was merely necessary for all trees and proved it was sufficient for regular ones. Then, the Rabin tree automaton for this condition accepted a superset of the tree models of the formula. But since the existence of any tree in this set implied the existence of a regular tree in it, the non-emptiness of this automaton's language was exactly equivalent to the satisfiability of the formula.

The heavy lifting in the first part of the thesis was in proving that our condition is indeed sufficient for regular trees. For this, we used combinatorial arguments to show that when the condition is not satisfied by a regular tree, it contains strict paths of unbounded lengths between two vertices, and is thus not embeddable. Showing that the condition is verifiable by a Rabin tree automaton was comparatively easier though by no means trivial, and was done by explicitly constructing the automata.

Our construction provided an ExpTime upper complexity bound, which is optimal since concept satisfiability w.r.t. general TBoxes is ExpTime-complete already for $\mathcal{ALC}$. Therefore we feel justified in taking an automata-theoretic approach for this problem, which allowed us to reveal that despite the fact that integer domains took quite long to be fully incorporated into description logics, they did not impose additional computational cost, which is surprising. Another aspect in which our automata-theoretic approach was justified was its applicability to related settings. We were able to adapt our constructions to a dense setting and facilitate a predicate for asserting that certain registers hold integer or natural numbers. This modeling capability has also been long sought-after, and came at no additional computational cost.

Finally, the first part was concluded with a more general discussion of description logics with concrete domains. We addressed the somewhat unorthodox syntax of $\mathcal{ALCF}^{\mathcal{P}}$ and compared it to the classical setting. We showed that some related logics can be translated into $\mathcal{ALCF}^{\mathcal{P}}$ using additional registers, which allowed us to obtain new results on DL-concrete domain combinations that were not previously considered in the literature.

The second part of the thesis was devoted to the conceptualization of a novel automata model tailored to a specific decidable logic with data values, namely $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$. As opposed to the logic in the first part, this logic already had tight complexity bounds, established by Schwentick and Zeume in [98]. Our Pebble-Intervals Automata were shown to capture this logic in the sense that for any definable class of data words, we could construct a PIA which accepts its string projection language. For this result, we first had to establish a normal form for the logic, reminiscent of the Scott Normal Form. We then had to develop numerous technical notions to navigate the transition from generic logical formulas, to the data words defined by them, to the constraints satisfied by these data words. Then we had to bridge the gap between there being no bound on the number of data values a data word can have, and there being a bound on the size of the state set of the PIA. For this, we had to develop notions which convey what information is

sufficient for determining the satisfaction of constraints, and then distill that information as we projected away the preorder to obtain strings that can be encoded into our PIA. These developments led to the second part of the thesis being heavy with technical detail, which was unfortunately unavoidable due to our manipulation of structures over different vocabularies.

We also provided some of the customary automata-model investigation by showing that PIA have a decidable emptiness test, and that they are closed under some common and not so common operators, such as shuffle. We showed that they are *not* effectively closed under intersection and complement, and concluded that their universality and inclusion problems are undecidable. In addition, we investigated which languages are accepted by PIA and saw that PI languages include all the regular languages, at least some context free and not context free languages, but certainly not all context sensitive languages.

In both parts, there was technical preprocessing in the form of normal forms and transformations. But the conditions the automata were used to verify were phrased in relatively natural terms. This allowed us to set aside the technical details for a moment and illustrate what can be expected from each logic in terms of expressiveness.

Further research

Both lines of research in this thesis have natural next steps. For our work on $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$, the main question is whether our technical developments can be transferred to other logics, for example to fragments [44] of constraint CTL*, or to stronger description logics such as $\mathcal{SHIQ}^{\mathcal{P}}(\mathcal{Z}_c)$. On the practical side, tableau algorithms are the most widely used tools for reasoning in expressive DLs, e.g. [100, 101, 102] and have enjoyed optimization efforts (see [53] and references in [103]), so it could be beneficial to develop a tableau algorithm for $\mathcal{ALCF}^{\mathcal{P}}(\mathcal{Z}_c)$. Nonetheless, some works have already developed practical approaches to automata construction for $\mathcal{ALCF}$ using alternating looping tree automata [104], and it would be interesting to extend these to the Rabin automata used here. Another avenue of future practical research would be to support reasoning with ABoxes and facilitating instance queries, as these are crucial for effective modeling and interaction with knowledge-based systems.

Regarding our results on Pebble-Intervals Automata, the most natural question would be whether PIA exactly capture $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ or are they stronger? Notice that our construction has the pebbles moving in one direction in each iteration, perhaps this behavior can be precisely formulated and the arising special case of PIA would exactly capture $\mathrm{FO}^2(\leq_1, \precsim_2, S_2)$ projection languages? Another natural question is whether our results can be extended to ω-languages. It *is* possible to extend our automata by a new type of transition which reads all positions with a specified letter between two pebbles in a single step while retaining the complexity of the emptiness problem, so it would be interesting to investigate the analog of Theorem 10.1 for infinite data words and their projections into ω-languages. Another important direction of research is the adaptation of the decidability results to trees, or to C^2, which extends FO^2 with counting

quantifiers [105]. Currently, the finite satisfiability problem is open for C^2 in the presence of a preorder, although it is known to be NExpTime-complete in the presence of a linear order [106]. Another related result is C^2 with an equivalence relation in forests [107], which is also NExpTime-complete.

Finally, it would also be interesting to explore the computational power of our automata model. We believe that the Dyck language of well-nested parentheses of two types may not be PI – can this be shown using a pumping lemma generalizing the ones for regular or context-free languages?

List of Figures

Bibliography

[1] J. McCarthy, M. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the Dartmouth Summer Research project on artificial intelligence, August 31, 1955," *AI Mag.*, vol. 27, no. 4, pp. 12–14, 2006.

[2] D. Hilbert and W. Ackerman, "Theoretische logik," *Julius Springer, Berlin*, 1928.

[3] K. Gödel, "Die vollständigkeit der axiome des logischen funktionenkalküls," *Monatshefte für Mathematik und Physik*, vol. 37, no. 1, pp. 349–360, 1930.

[4] A. Church, "A note on the Entscheidungsproblem," *J. Symb. Log.*, vol. 1, no. 1, pp. 40–41, 1936.

[5] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.

[6] B. Trakhtenbrot, "The impossibility of an algorithm for the decidability problem on finite classes," in *Proceedings of the USSR Academy of Sciences*, vol. 70, pp. 569–572, 1950.

[7] E. Grädel, "Satisfiability of formulae with one $\forall$ is decidable in exponential time," *Arch. Math. Log.*, vol. 29, no. 4, pp. 265–276, 1990.

[8] A. S. Kahr, E. F. Moore, and H. Wang, "Entscheidungsproblem reduced to the AEA case," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 48, no. 3, p. 365, 1962.

[9] E. Börger, E. Grädel, and Y. Gurevich, *The Classical Decision Problem*. Perspectives in Mathematical Logic, Springer, 1997.

[10] H. Andréka, I. Németi, and J. van Benthem, "Modal languages and bounded fragments of predicate logic," *J. Philos. Log.*, vol. 27, no. 3, pp. 217–274, 1998.

[11] E. Grädel, "On the restraining power of guards," *J. Symb. Log.*, vol. 64, no. 4, pp. 1719–1742, 1999.

[12] M. Mortimer, "On languages with two variables," *Mathematical Logic Quarterly*, vol. 21, no. 1, pp. 135–140, 1975.

[13] E. Grädel, P. G. Kolaitis, and M. Y. Vardi, "On the decision problem for two-variable first-order logic," *Bulletin of symbolic logic*, vol. 3, no. 01, pp. 53–69, 1997.

[14] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David, "Two-variable logic on words with data," in *LICS*, pp. 7–16, IEEE, 2006.

[15] E. Kieronski and L. Tendera, "On finite satisfiability of two-variable first-order logic with equivalence relations," in *LICS*, pp. 123–132, IEEE Computer Society, 2009.

[16] D. Calvanese, T. Kotek, M. Šimkus, H. Veith, and F. Zuleger, "Shape and content," in *Integrated Formal Methods*, pp. 3–17, Springer, 2014.

[17] D. Calvanese, M. Ortiz, and M. Simkus, "Verification of evolving graph-structured data under expressive path constraints," in *ICDT*, pp. 15:1–15:19, 2016.

[18] A. Rensink, "Canonical graph shapes," in *Programming Languages and Systems*, vol. 2986 of *LNCS*, pp. 401–415, Springer, 2004.

[19] R. M. Nowak, *Walker's marsupials of the world.* JHU Press, 2005.

[20] F. Baader and P. Hanschke, "A scheme for integrating concrete domains into concept languages," in *Proc. of IJCAI 1991*, pp. 452–457, Morgan Kaufmann, 1991.

[21] C. Lutz, "PSpace reasoning with the description logic ALCF(D)," *Logic Journal of the IGPL*, vol. 10, no. 5, pp. 535–568, 2002.

[22] C. Lutz, "NEXPTIME-complete description logics with concrete domains," *ACM Trans. Comput. Logic*, vol. 5, no. 4, p. 669–705, 2004.

[23] C. Lutz, "Description logics with concrete domains-a survey," in *Proc. of Advances in Modal Logic 4*, pp. 265–296, King's College Publications, 2002.

[24] M. Otto, "Two variable first-order logic over ordered domains," *Journal of Symbolic Logic*, pp. 685–702, 2001.

[25] K. Etessami, M. Y. Vardi, and T. Wilke, "First-order logic with two variables and unary temporal logic," *Information and computation*, vol. 179, no. 2, pp. 279–295, 2002.

[26] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin, "Two-variable logic on data words," *TOCL*, vol. 12, no. 4, p. 27, 2011.

[27] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin, "Two-variable logic on data trees and XML reasoning," in *PODS*, pp. 10–19, ACM, 2006.

[28] C. David, L. Libkin, and T. Tan, "On the satisfiability of two-variable logic over data words," in *LPAR*, vol. 6397, pp. 248–262, 2010.

[29] M. Niewerth and T. Schwentick, "Two-variable logic and key constraints on data words," in *ICDT*, pp. 138–149, ACM, 2011.

[30] M. Schützenberger, "On finite monoids having only trivial subgroups," *Information and Control*, vol. 8, no. 2, pp. 190–194, 1965.

[31] R. McNaughton and S. A. Papert, *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.

[32] C. C. Elgot, "Decision problems of finite automata design and related arithmetics," *Transactions of the American Mathematical Society*, vol. 98, no. 1, pp. 21–51, 1961.

[33] J. R. Büchi, "Weak second-order arithmetic and finite automata," *Z. Math. Logik Grundl. Math.*, vol. 6, pp. 66–92, 1960.

[34] B. A. Trakhtenbrot, "Finite automata and the logic of monadic predicates," in *Doklady Akademii Nauk*, vol. 140, pp. 326–329, Russian Academy of Sciences, 1961.

[35] B. Courcelle and J. Engelfriet, *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, vol. 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012.

[36] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, eds., *Handbook of Model Checking*. Springer International Publishing, 2018.

[37] D. Calvanese, G. D. Giacomo, and M. Lenzerini, "2ATAs make DLs easy," in *Description Logics*, vol. 53 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2002.

[38] C. Carapelle and A. Turhan, "Description logics reasoning w.r.t. general TBoxes is decidable for concrete domains with the EHD-property," in *Proc. of ECAI 2016*, vol. 285 of *Frontiers in Artificial Intelligence and Applications*, pp. 1440–1448, IOS Press, 2016.

[39] F. Baader and P. Hanschke, "Extensions of concept languages for a mechanical engineering application," in *GWAI*, vol. 671 of *LNCS*, pp. 132–143, Springer, 1992.

[40] C. Lutz, "NEXPTIME-complete description logics with concrete domains," in *Proc. of IJCAR 2001*, vol. 2083 of *LNCS*, pp. 45–60, Springer, 2001.

[41] F. Baader and J. Rydval, "Description logics with concrete domains and general concept inclusions revisited," in *IJCAR (1)*, vol. 12166 of *Lecture Notes in Computer Science*, pp. 413–431, Springer, 2020.

[42] C. Lutz and M. Milicic, "A tableau algorithm for description logics with concrete domains and general TBoxes," *J. Autom. Reasoning*, vol. 38, no. 1-3, pp. 227–259, 2007.

[43] C. Lutz, "Adding numbers to the SHIQ description logic: First results," in *Proc. of KR 2002*, pp. 191–202, 2002.

[44] L. Bozzelli and R. Gascon, "Branching-time temporal logic extended with qualitative Presburger constraints," in *Proc. of LPAR 2006*, vol. 4246 of *LNCS*, pp. 197–211, Springer, 2006.

[45] C. Carapelle and A.-Y. Turhan, "Description logics reasoning w.r.t. general TBoxes is decidable for concrete domains with the EHD-property," LTCS-Report 16-01, Chair of Automata Theory, TU Dresden, 2016.

[46] T. Schwentick and T. Zeume, "Two-variable logic with two order relations," in *CSL*, pp. 499–513, Springer, 2010.

[47] P. Brass, *Advanced data structures*, vol. 193. Cambridge University Press Cambridge, 2008.

[48] W. Thomas, "Automata on infinite objects," in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pp. 133–191, Elsevier and MIT Press, 1990.

[49] R. Diestel, *Graph Theory, 4th Edition*, vol. 173 of *Graduate texts in mathematics*. Springer, 2012.

[50] M. Sipser, *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[51] S. Arora and B. Barak, *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[52] W. J. Savitch, "Relationships between nondeterministic and deterministic tape complexities," *J. Comput. Syst. Sci.*, vol. 4, no. 2, pp. 177–192, 1970.

[53] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds., *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.

[54] M. O. Rabin, "Decidability of second-order theories and automata on infinite trees," *Transactions of the American Mathematical Society*, vol. 141, pp. 1–35, 1969.

[55] M. Schmidt-Schauß and G. Smolka, "Attributive concept descriptions with complements," *Artif. Intell.*, vol. 48, no. 1, pp. 1–26, 1991.

[56] K. Schild, "A correspondence theory for terminological logics: Preliminary report," in *Proc. of IJCAI 1991*, pp. 466–471, Morgan Kaufmann, 1991.

[57] K. Schild, "Terminological cycles and the propositional μ-calculus," in *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94). Bonn, Germany, May 24-27, 1994* (J. Doyle, E. Sandewall, and P. Torasso, eds.), pp. 509–520, Morgan Kaufmann, 1994.

[58] C. Lutz, *The complexity of description logics with concrete domains*. PhD thesis, RWTH Aachen University, Germany, 2002.

[59] B. Hollunder, W. Nutt, and M. Schmidt-Schauß, "Subsumption algorithms for concept description languages," in *9th European Conference on Artificial Intelligence, ECAI 1990, Stockholm, Sweden, 1990*, pp. 348–353, 1990.

[60] G. D. Giacomo, *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Universitá de Roma "La Sapienza", 1995.

[61] S. Tobies, *Complexity results and practical algorithms for logics in knowledge representation*. PhD thesis, RWTH Aachen University, Germany, 2001.

[62] M. Y. Vardi, "Why is modal logic so robustly decidable?," in *Descriptive Complexity and Finite Models*, vol. 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 149–183, DIMACS/AMS, 1996.

[63] E. A. Emerson and C. S. Jutla, "The complexity of tree automata and logics of programs (extended abstract)," in *Proc. of FOCS 1988*, pp. 328–337, IEEE Computer Society, 1988.

[64] D. E. Muller and P. E. Schupp, "Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra," *Theor. Comput. Sci.*, vol. 141, no. 1&2, pp. 69–107, 1995.

[65] M. O. Rabin, *Automata on Infinite Objects and Church's Problem*. Boston, MA, USA: American Mathematical Society, 1972.

[66] F. Baader, "Description logics," in *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School*, vol. 5689 of *LNCS*, pp. 1–39, Springer, 2009.

[67] M. Bojańczyk and S. Toruńczyk, "Weak MSO+U over infinite trees," in *Proc. of STACS 2012*, vol. 14 of *LIPIcs*, pp. 648–660, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[68] C. Carapelle, A. Kartzow, and M. Lohrey, "Satisfiability of ECTL* with constraints," *J. Comput. Syst. Sci.*, vol. 82, no. 5, pp. 826–855, 2016.

[69] S. Demri and D. D'Souza, "An automata-theoretic approach to constraint LTL," *Inf. Comput.*, vol. 205, no. 3, pp. 380–415, 2007.

[70] C. Carapelle, A. Kartzow, and M. Lohrey, "Satisfiability of CTL* with constraints," in *Proceedings of CONCUR 2013*, vol. 8052 of *LNCS*, pp. 455–469, Springer, 2013.

[71] C. Lutz, "Combining interval-based temporal reasoning with general TBoxes," *Artif. Intell.*, vol. 152, no. 2, pp. 235–274, 2004.

[72] F. Baader, H. Bürckert, B. Nebel, W. Nutt, and G. Smolka, "On the expressivity of feature logics with negation, functional uncertainty, and sort equations," *Journal of Logic, Language and Information*, vol. 2, no. 1, pp. 1–18, 1993.

[73] C. Lutz, "Complexity of terminological reasoning revisited," in *Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, vol. 1705 of *Lecture Notes in Computer Science*, pp. 181–200, Springer, 1999.

[74] G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*. Springer, 1997.

[75] M. Moortgat, "A note on multidimensional Dyck languages," in *Categories and Types in Logic, Language, and Physics*, vol. 8222 of *Lecture Notes in Computer Science*, pp. 279–296, Springer, 2014.

[76] M. L. Minsky, *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

[77] R. Schroeppel, "A two counter machine cannot calculate 2^N," tech. rep., A.I. Laboratory, Massachusetts Institute of Technology, 1972.

[78] A. Meduna and P. Zemek, "Jumping finite automata," *Int. J. Found. Comput. Sci.*, vol. 23, no. 7, pp. 1555–1578, 2012.

[79] H. Fernau, M. Paramasivan, M. L. Schmid, and V. Vorel, "Characterization and complexity results on jumping finite automata," *Theor. Comput. Sci.*, vol. 679, pp. 31–52, 2017.

[80] M. Mortimer, "On languages with two variables," *Math. Log. Q.*, vol. 21, no. 1, pp. 135–140, 1975.

[81] E. Grädel, P. G. Kolaitis, and M. Y. Vardi, "On the decision problem for two-variable first-order logic," *Bulletin of symbolic logic*, vol. 3, no. 01, pp. 53–69, 1997.

[82] M. Fürer, "The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems)," in *Logic and Machines*, vol. 171 of *Lecture Notes in Computer Science*, pp. 312–319, Springer, 1983.

[83] D. Scott, "A decision method for validity of sentences in two variables," *Journal of Symbolic Logic*, vol. 27, no. 377, p. 74, 1962.

[84] E. Grädel, M. Otto, and E. Rosen, "Undecidability results on two-variable logics," *Arch. Math. Log.*, vol. 38, no. 4-5, pp. 313–354, 1999.

[85] E. Kieronski, "Results on the guarded fragment with equivalence or transitive relations," in *CSL*, vol. 3634 of *Lecture Notes in Computer Science*, pp. 309–324, Springer, 2005.

[86] W. Szwast and L. Tendera, "FO^2 with one transitive relation is decidable," in *STACS*, vol. 20 of *LIPIcs*, pp. 317–328, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[87] I. Pratt-Hartmann, "Finite satisfiability for two-variable, first-order logic with one transitive relation is decidable," *Math. Log. Q.*, vol. 64, no. 3, pp. 218–248, 2018.

[88] T. Zeume and F. Harwath, "Order-invariance of two-variable logic is decidable," in *LICS*, pp. 807–816, ACM, 2016.

[89] E. Kieronski, "Decidability issues for two-variable logics with several linear orders," in *CSL*, vol. 12 of *LIPIcs*, pp. 337–351, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[90] M. Kaminski and N. Francez, "Finite-memory automata," *Theor. Comput. Sci.*, vol. 134, no. 2, pp. 329–363, 1994.

[91] F. Neven, T. Schwentick, and V. Vianu, "Finite state machines for strings over infinite alphabets," *TOCL*, vol. 5, no. 3, pp. 403–435, 2004.

[92] P. Bouyer, A. Petit, and D. Thérien, "An algebraic approach to data languages and timed languages," *Inf. Comput.*, vol. 182, no. 2, pp. 137–162, 2003.

[93] J. Esparza, "Decidability and complexity of petri net problems - an introduction," in *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, pp. 374–428, 1996.

[94] H. Björklund and T. Schwentick, "On notions of regularity for data languages," *Theor. Comput. Sci.*, vol. 411, no. 4-5, pp. 702–715, 2010.

[95] O. Grumberg, O. Kupferman, and S. Sheinvald, "Variable automata over infinite alphabets," in *LATA*, vol. 6031, pp. 561–572, Springer, 2010.

[96] L. Segoufin, "Automata and logics for words and trees over an infinite alphabet," in *CSL*, pp. 41–57, Springer, 2006.

[97] A. Kara, *Logics on Data Words*. PhD thesis, Technical University of Dortmund, 2016.

[98] T. Zeume and T. Schwentick, "Two-variable logic with two order relations," *Log. Meth. Comput. Sci.*, vol. 8, 2012.

[99] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *POPL*, pp. 238–252, 1977.

[100] D. Tsarkov and I. Horrocks, "FaCT++ description logic reasoner: System description," in *Automated reasoning*, pp. 292–297, Springer, 2006.

[101] E. Sirin and B. Parsia, "Pellet system description," in *Description Logics*, vol. 189 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2006.

[102] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: An OWL 2 reasoner," *J. Autom. Reasoning*, vol. 53, no. 3, pp. 245–269, 2014.

[103] D. Tsarkov, I. Horrocks, and P. F. Patel-Schneider, "Optimizing terminological reasoning for expressive description logics," *Journal of Automated Reasoning*, vol. 39, no. 3, pp. 277–316, 2007.

[104] D. Calvanese, D. Carbotta, and M. Ortiz, "A practical automata-based technique for reasoning in expressive description logics," in *IJCAI*, pp. 798–804, IJCAI/AAAI, 2011.

[105] E. Grädel, M. Otto, and E. Rosen, "Two-variable logic with counting is decidable," in *LICS*, pp. 306–317, IEEE Computer Society, 1997.

[106] W. Charatonik and P. Witkowski, "Two-variable logic with counting and a linear order," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 41, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[107] W. Charatonik, Y. Guskov, I. Pratt-Hartmann, and P. Witkowski, "Two-variable first-order logic with counting in forests," in *LPAR*, vol. 57 of *EPiC Series in Computing*, pp. 214–232, EasyChair, 2018.