A Journey from Finite to Automatic Structures and Beyond

A Dissertation
Submitted to the Department of Computer Science
and the School of Graduate Studies
of University of Auckland
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY

Jiamou Liu
November 2010

Preface

The work at hand studies properties of structures that have certain types of algorithmic description. In particular, we study the fundamental properties of three classes of structures. The first class is the class of finite structures, which can be algorithmically described by listing all elements and tuples of atomic relations. The second class is the class of automatic structures, which are possibly infinite structures whose descriptions consist of automata representing their universe and relations. These structures attract attention as their first-order theories are decidable. The third class is the class of computable structures, which are structures given by Turing machines.

For finite structures, we address the efficiency of deciding the winners in Ehrenfeucht-Fraïssé games (EF games for short) for some standard classes of structures. EF game is an important tool in finite model theory in demonstrating the expressive power of first-order logic and its extensions. We present algorithms that decide which player wins EF games played on structures that are taken from the following classes: structures with unary predicates, equivalence structures and some of their expansions, trees with level predicates and Boolean algebras with distinguished ideals. Under some natural assumptions on the representations of these structures and EF games, we will prove all algorithms run in constant time.

We then investigate a special subclass of automatic structures, the class of unary automatic structures. These structures are described using automata over a unary alphabet. We present uniform and efficient algorithms to decide certain graph-theoretical properties for the class of unary automatic graphs with finite degree. We also provide efficient algorithms for deciding the isomorphism problem for unary automatic linear orders, equivalence structures and trees. We also extend the notion of state complexity from regular languages to structures and study this in the context of unary automatic structures.

For automatic structures in general, the isomorphism problem is highly undecidable (Σ^1_1-complete). We show that undecidability also holds for some natural subclasses of automatic structures. In particular, we show that the isomorphism problem for automatic equivalence structures is Π^0_1-complete; the isomorphism problem for automatic successor trees of finite height $k \geq 2$ is Π^0_{2k-3}-complete; the isomorphism problem for automatic linear

orders is hard for every level of the arithmetic hierarchy. We also illustrate that for any $k \in \mathbb{N}$, there exist two isomorphic automatic trees of finite height (and two automatic linear orders) without any Σ^0_k-isomorphism. These solve some known open questions in the area, in particular, questions posed by Khoussainov and Nerode.

Lastly, we study computable categoricity of computable structures. A computable structure is computably categorical if any two computable presentation of it are computably isomorphic. We focus on the class of computably categorical graphs. In particular, we investigate the strongly locally finite graphs: graphs all of whose components are finite. We present a necessary and sufficient condition for certain classes of strongly locally finite graphs to be computably categorical. We show that whenever the graph contains an infinite Δ^0_2-set of components that can be properly embedded into infinitely many components of the graph, then the graph is not computably categorical. We also construct a strongly locally finite computably categorical graph with an infinite chain of properly embedded components.

Acknowledgements

I owe my deepest gratitude to my main supervisor, Bakhadyr Khoussainov, whose instructions, encouragement and support have made great impact in my development. I thank him for introducing me to the subject, suggesting me interesting problems and teaching me important techniques in my research. Through his association, I have had the privilege to work with some distinguished people. I am indebted to all of the co-authors who provided me great help on the work in this thesis: Barbara Csima, Bakhadyr Khoussainov, Dietrich Kuske, Markus Lohrey and Mia Minnes. In particular, I am very grateful to Markus Lohrey, who hosted me at Leipzig. In Leipzig I was fortunate to work with Markus and Dietrich Kuske on automatic structures. I have learned a lot from both of them.

I want to thank André Nies for helping me on computability theory and providing me interesting ideas to expand my research.

I want to thank Richard Shore and Anil Nerode for hosting me at Cornell University in the Fall semesters of 2005, 2007 and 2008. The courses I took from them helped me immensely in understanding mathematical logic.

I also want to thank Ting Zhang for being my mentor during my internship at Microsoft Research Asia, Beijing and Wei Chen for teaching me distributed computing at Tsinghua University. Many thanks to Yang Yue who organized my three-month visit to National University of Singapore in 2006.

I want to thank Education New Zealand for providing me funding for three years through NZIDRS (New Zealand International Doctoral Research Scholarship). I am also indebted to Microsoft Research Asia for financial support through Microsoft Research Asia Fellowship.

Also, I would like to thank Mingzhong Cai, Aniruddh Gandhi, Niko Haubold, Nick Hay, Masoud Khosravani, Pavel Semukhin, Christian Mathissen, Alexander Melnikov, Sasha Rubin and all friends and colleagues who provided a stimulating work environment during my stays in different institutions during the past years.

Above all, I thank my family. This thesis would not have been possible without the continuous encouragement and support from my parents, Xu Jin and Liu Mingdi. Jade, thank you for your understanding, patience and love during the past 4 years. Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and motivation

The goal of the thesis is to study properties of structures that have certain types of algorithmic descriptions. Such structures have attracted the attention of many experts in mathematical logic and algebra. These structures have also become a topic of interest to experts in theoretical computer science, especially to those in computational complexity, model checking, verification, and logic in computer science. By a structure, we mean a set along with a collection of finitary functions and relations defined on it. In mathematics, typical examples of structures are orders, lattices, Boolean algebras, groups, rings, fields and vector spaces. In computer science, structures with no functions represent models of relational databases. Structures are also used to realize specifications of systems on which formal verifications are carried out. Broadly speaking, data structures such as tables, lists and trees can also be viewed as structures. Other examples of structures in computer science include models of XML documents, programs and networks.

In the thesis, we study some of the basic yet fundamental properties of three classes of structures with algorithmic descriptions. The first class is the class of *finite structures*. These structures have obvious algorithmic descriptions that consist of listing all domain elements and tuples of atomic relations. The second class is the class of *automatic structures*. These structures are typically infinite but their descriptions consist of automata representing their domain and relations. Examples of such structures include Presburger arithemetic $(\mathbb{N}; +)$ and the Skolem arithmetic $(\mathbb{N}; \times)$. The third class is the class of *computable structures*. The descriptions of structures from this class consist of Turing machines. Typically, the domains of these structures are computable subsets of natural numbers and all atomic relations are uniformly computable. The arithmetic $(\mathbb{N}; +, \times, \leq, 0)$ is an example of a computable structure.

This thesis contains a collection of results on finite structures, automatic structures and

computable structures. In particular, these results aim to answer the following interrelated problems.

- *Problem 1. How complex is it to compute whether two structures are similar?*
- *Problem 2. How complex is it to compute whether two structures are isomorphic?*
- *Problem 3. How complex is it to build an isomorphism between structures?*

All these problems assume certain explicit presentations of the underlying structures. The unifying motivation of these problems is to understand the complexities, in various senses, of classes of structures that are respectively finite, automatic, and computable.

1.1.1 Finite model theory

The central themes of finite model theory concern with the expressibility of logics and its connection with computational complexity. The first investigation into the properties of logical languages over finite structures probably dates back to Trakhtenbrot in 1950 [113], who proved validity over finite structures is not computably enumerable. Since then finite model theory has evolved into a separate field from classical model theory. This is largely due to the fact that most tools in classical model theory, e.g. compactness theorem, Łoś-Tarski theorem [110], Craig interpolation theorem [47], do not have counterparts over finite structures. On the other hand, Ehrenfeucht-Fraïssé games, a notion that is already present in infinite model theory, have become a central technique in finite model theory.

The Ehrenfeucht-Fraïssé games were used in establishing various inexpressibility results in first order logic. See Gurevich [46] for typical examples. Building on classical work by Hanf [48] and Gaifman [31], Fagin/Stockmeyer/Vardi [27], Schwentick [105], Arora/Fagin [2] and Hella/Libkin/Nurmonen [51] provided sufficient winning condition for Ehrenfeucht-Fraïssé games. Variants of Ehrenfeucht-Fraïssé games were also used to prove expressibility and inexpressibility results in other logics. Examples along this line of research include Fagin [26] and Ajtai/Fagin [1] on existential monadic second order logic; Hella [50] on infinitary counting logics; Immerman [60] and Poizat [95] on logics with finitely many variables.

Descriptive complexity established the connections between logic and computational complexity. The celebrated result by Fagin [25] showed that properties that are decidable in nondeterministic polynomial times correspond exactly to the ones definable in existential second order logic. Over ordered structures, Immerman [59] and Vardi [116] showed that least fixed-point logic captures the class of polynomial time decidable properties and Immerman [61] showed that transitive closure logic captures the class of logarithmic space decidable properties.

We refer the readers to standard textbooks of Ebbinghaus/Flum [19] and Libkin [85], as well as the book Grädel et al. [42] for detailed accounts of finite model theory.

1.1.2 Automatic structures

The idea of automatic structures goes back to Büchi [10] and Elgot [21] who established equivalence between monadic second order logic and finite automata. The result was used to decide S1S, the monadic second order logic of the natural numbers with one successor relation. Later Rabin [98] used automata on infinite trees to decide S2S, the monadic second order logic of the infinite binary trees with two successor predicates. Hodgson [55] first introduced the term "automata decidable theory". In 1995, Khoussainov/Nerode [70] introduced automata presentable structures as part of the complexity theoretic model theory, and initiated a systematic study of automatic structures.

Roughly, we say that a relational structure is automatic if the elements in the universe can be represented as strings from a regular language and every relation of the structure can be recognized by a finite state automaton with several heads that proceed synchronously. The representation of an automatic structure is the collection of automata that recognize respectively its domain and relations, and is therefore finite. The class of automatic structures are closed under the logical operations $\vee$, $\neg$ and $\exists$. Hence, one can effectively decide any properties that are defined by first order logic for these structures. Therefore, automatic structures fit into the program of algorithmic model theory, which focuses on structures that have both finite presentations and effective semantics.

Algorithmic model theory is motivated by applications where infinite structures (e.g. in databases or program verifications [118, 117, 45]) are of interest. A wide range of finitely presentable structures have been investigated in the past, which include, apart from automatic structures, structures that are definable through graph grammars, or through interpretations over a fixed structure. See the recent survey [4] for an exposition of algorithmic model theory.

Numerous works have focused on the logical properties of automatic structures. The first order decidability result mentioned above has been extended by adding the quantifiers $\exists^\infty$ ("there exists infinitely many") [6], $\exists^{(k,m)}$ ("there exists k modulo m many") [76], and a number of other generalized quantifiers [103]. Blumensath/Grädel [8] proved a logical characterization theorem stating that finite-word automatic structures are exactly those definable in the following fragment of the arithmetic $(\mathbb{N}; +, |_2, \leq, 0)$, where $+$ and $\leq$ have their usual meanings and $|_2$ is a weak divisibility predicate for which $x|_2 y$ if and only if x is a power of 2 and divides y.

Different concepts and tools were employed to identify structures that are not automatic. For example, techniques in automata theory such as the pumping lemma were used to prove that $(\mathbb{N}; \times)$ is not automatic [8]. Some more combinatorial and model-theoretical arguments were used to prove e.g. that the random graph is not automatic [72, 16]. Recently, Tsankov [114] used advanced techniques from additive combinatorics to prove that $(\mathbb{Q}; +)$, the additive group of the rationals, is not automatic. This illustrates the complexity and depth

of research in automatic structures.

Attentions have also turned to characterizing specific subclasses of automatic structures. There are descriptions of automatic linear orders and trees in terms of model theoretic concepts such as Cantor-Bendixson ranks [75]. Also, [72] characterised the isomorphism types of automatic Boolean algebras; Thomas/Oliver [93] gave a full description of finitely generated automatic groups using the famous theorem of Gromov about finitely generated groups with polynomial growth. Some of these results have direct algorithmic implications. For example, the isomorphism problem for automatic well-ordered sets and Boolean algebras is decidable.

Most of the results concerning automatic structures, including the ones mentioned above, demonstrate that in various concrete senses automatic structures are not complex. However, this intuition can be misleading. For example, Khoussainov/Nies/Rubin/Stephan in [72] showed that the isomorphism problem for automatic structures is Σ^1_1-complete. This tells us informally that there is no hope for a description (in a natural logical language) of the isomorphism types of automatic structures. Also, Khoussainov/Minnes in [69] provided examples of automatic structures whose Scott ranks can be as high as possible, fully covering the interval $[1, \omega_1^{CK} + 1]$ of ordinals (where ω_1^{CK} is the first non-computable ordinal). They also showed that the ordinal heights of well-founded automatic relations can be arbitrarily large ordinals below ω_1^{CK}.

For introduction and overview of automatic structures, we refer the readers to the theses of Blumensath [6], Rubin [102], Bárány [3], Minnes [91] as well as the survey papers Khoussainov/Minnes [68], Nies [92] and Rubin [103]. We also mention that there is an important body of work on structures presented by variants of finite automata such as the infinite word (Büchi) automata, and finite/infinite tree automata. See Benedikt/Libkin/Neven [5], Colcombet [13], Kuske/Lohrey [83], Hjorth/Khoussainov/Montalbán/Nies [53] and Kaiser/Rubin/Bárány [62]. The algorithmic and logical properties of these alternative forms of automatic structures are relatively less known compared with the finite-word counterparts and is beyond the scope of the work at hand.

1.1.3 Computable model theory

A structure is computable if its domain and all relations are computable sets of natural numbers. The earliest accounts on computable structures trace back to van der Waerden [115] and Frölich/Shepherdson [29, 30]. Systematic studies on computable structures started in the 1960s by Rabin [96, 97] and Mal'cev [87]. This is followed by the works of Ershov [22], Goncharov [37, 38, 39, 40], Metakides/Nerode [89] that lay out the foundation of the nowadays well-developed computable model theory.

Computable model theory seeks to capture the effective content of model-theoretic

constructions and results. Typical topics include constructing models of first order theories such as prime, homogeneous and saturated models, building isomorphisms between computable structures, studying the degree spectra of relations, and understanding the relationship between definability and computability. Techniques in computability theory have often been used in computable model theory. Such techniques include priority argument with finite and infinite injuries and constructions that are put on trees. The reader is referred to *Handbooks of recursive mathematics* [23], *Handbook of computability theory* [43] and the papers [79, 35, 17, 90] for introduction into this exciting field.

1.2 Summary of results

In the following, we present the topics and results obtained in each chapter of the thesis. Formal definitions and proofs are contained in the corresponding chapters.

Chapter 2. Preliminaries

This chapter presents a necessary background to model theory and computability theory needed throughout the thesis. In particular, the chapter introduces automatic and computable structures. In this thesis, we assume that all structures are countable and relational (that is, have no function symbols in their language). We always make this assumption since functions can be replaced by their graphs.

Definition 2.5.1. A structure is called *automatic* if its domain is a regular language and all relations are recognized by synchronous multi-tape automata.

Definition 2.5.10. A structure is called *computable* if its domain is a computable subset of natural numbers and all its relations are uniformly computable.

Consider $\mathsf{FO} + \exists^{\infty} + \exists^{n,n}$, the first-order logic extended by the quantifiers $\exists^{\infty}$ and $\exists^{n,m}$. The following theorem from [55, 70, 7] is one of the main motivations for investigating automatic structures.

Theorem 2.5.11. For an automatic structure $\mathcal{A}$, there is an algorithm that, given a formula $\varphi(\overline{x})$ in $\mathsf{FO} + \exists^{\infty} + \exists^{n,m}$, produces an automaton whose language consists of those tuples $\overline{a}$ from $\mathcal{A}$ such that $\mathcal{A} \models \varphi(\overline{a})$. Hence, the $\mathsf{FO} + \exists^{\infty} + \exists^{n,m}$ theory of any automatic structure is decidable.

Chapter 3. The complexity of Ehrenfeucht-Fraïssé games

This chapter focuses on finite structures. We address the efficiency of deciding the winners of Ehrenfeucht-Fraïssé games for some standard classes of finite structures. An Ehrenfeucht-Fraïssé game (EF game for short) is a two-player game played on two structures $\mathcal{A}$ and $\mathcal{B}$ of the same signature. We call the two players of the game respectively Spoiler and Duplicator. For a natural number $n \in \mathbb{N}$, the *n-round EF-game on $\mathcal{A}$ and $\mathcal{B}$*, denoted by $G_n(\mathcal{A}, \mathcal{B})$, is played by the two players moving in *n rounds*. At each round, Spoiler selects structure $\mathcal{A}$ or $\mathcal{B}$, and then selects an element from the selected structure. Then, Duplicator responds by selecting an element from the other structure. Hence over a sequence of n rounds, the players produce a sequence $a_1, \ldots, a_n$ of elements in $\mathcal{A}$ and a sequence $b_1, \ldots, b_n$ of elements in $\mathcal{B}$ such that for $1 \leq i \leq n$, (a_i, b_i) is the pair of elements selected by the players in round i. Duplicator *wins* the play if the mapping $a_i \to b_i, i = 1, \ldots, n$, is a partial isomorphism between $\mathcal{A}$ and $\mathcal{B}$. Duplicator *wins* the game $G_n(\mathcal{A}, \mathcal{B})$ if she can always select elements in a way that wins the play regardless of the sequence of elements that Spoiler selects.

Informally, Duplicator's goal is to show that the two structures $\mathcal{A}$ and $\mathcal{B}$ are "similar", while Spoiler needs to show the opposite. It is clear that when $\mathcal{A}$ and $\mathcal{B}$ are isomorphic, Duplicator wins $G_n(\mathcal{A}, \mathcal{B})$ for all $n \in \mathbb{N}$. On the other hand, when $\mathcal{A}$ and $\mathcal{B}$ are finite structures, for large n (where n is greater or equal to the largest cardinality of $\mathcal{A}$ and $\mathcal{B}$), if Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ then $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. It is known that for all $n \in \mathbb{N}$, Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ if and only if $\mathcal{A}$ and $\mathcal{B}$ satisfy the same first order formulas of quantifier rank n [28, 20]. Hence, these games can be viewed as a way for approximating if two structures are isomorphic.

It is thus interesting to develop tools and algorithms for finding winners of EF games. Grohe [44] studied EF games with a fixed number of pebbles and showed that the problem of deciding the winner is PTIME-complete. Pezzoili [94] showed that deciding the winner of EF games is PSPACE-complete. Kolaitis/Panttaja [80] proved that the following problem is EXPTIME-complete: given a natural number k and structures $\mathcal{A}$ and $\mathcal{B}$, decide the winner for the k pebble existential EF game on $\mathcal{A}$ and $\mathcal{B}$.

Fix a natural number $n \in \mathbb{N}$. We concern the following question that we call the *n-Ehrenfeucht-Fraïssé* problem.

INPUT: Two relational structures $\mathcal{A}$ and $\mathcal{B}$ from a fixed class of structures
QUESTION: Does Duplicator win the n-round EF game $G_n(\mathcal{A}, \mathcal{B})$?

In this chapter, we solve the Ehrenfeucht-Fraïssé problem for the following classes of finite structures:

1. structures with only unary predicates

2. equivalence structures and their extensions

3. trees with height predicates

4. Boolean algebras with distinguished ideals

We provide algorithms for solving the Ehrenfeucht-Fraïssé problem for the structures mentioned above. The running time of all the algorithms are bounded by constants. We obtain the values of these constants as functions of n. As an example, we briefly describe our result for equivalence structures, which are structures of the form $(D; E)$ where E is an equivalence relation. For any EF game played on two equivalence structures, we define two conditions, small disparity and large disparity, each of which guarantees winning for Spoiler. We define these conditions using the numbers of equivalence classes of some particular sizes in both structures. We then prove that these conditions are necessary for Spoiler to win the EF game. To do that, assuming neither small nor large disparity occurs, we describe a strategy for Duplicator that ensures all plays satisfy some invariants at all rounds of the game. In particular, these invariants imply that the strategy is winning for Duplicator. Hence, to compute the winner of an EF game played on equivalence structures, it suffices to check if either small or large disparity occurs, which can be done in constant time under some assumptions on the representations of the structures. As a result, we obtain the following theorem:

Theorem 3.3.5. Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ on finite equivalence structures $\mathcal{A}$ and $\mathcal{B}$. The constant that bounds the running time is n.

We then extend the above technique to variants of equivalence structures. For example, an *equivalence structure with s colors* is a structures of the type $(A; E, P_1, \ldots, P_s)$, where E is an equivalence relation on A and $P_1, \ldots, P_s$ are unary predicates. An *embedded equivalence structure of height h* is the form $\mathcal{A} = (A; E_1, E_2, ..., E_h)$ such that each E_i where $1 \leq i \leq h$ is an equivalence relation and $E_i \subseteq E_j$ for $i < j$. For these extended notions of equivalence structures, we define different forms of disparities and prove that they are necessary and sufficient conditions for Spoiler to win the EF game.

Theorem 3.4.10. Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins the n-round Ehrenfeucht-Fraïssé game $G_n(\mathcal{A}, \mathcal{B})$ on finite equivalence structures with s colors. The constant that bounds the running time is n^{2^s+1}.

Theorem 3.5.6. Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins game $G_n(\mathcal{A}, \mathcal{B})$ on finite embedded equivalence structures of height h $\mathcal{A} = (A; E_1, ..., E_h)$ and $\mathcal{B} = (B; E_1, ..., E_h)$. The constant that bounds the running time is $< (n+1)^{\cdots^{(n+1)^{(n+1)}}}$ where the tower of $(n+1)$ has height h.

Table 1.1: Deciding the EF games on classes of finite structures

Classes of finite structure	Time bound for EF games
Structures with s unary predicates	$2^s \cdot n$
Equivalence structures	n
Homogeneously s-colored equivalence structures	$2^s \cdot n$
Equivalence structures with s colors	n^{2^s+1}
Embedded equivalence structures of height h	height h tower $(n+1)^{\cdots^{(n+1)^{(n+1)}}}$
Trees with level predicates of height h	height h tower $(n+1)^{\cdots^{(n+1)^{(n+1)}}}$
Boolean algebras with s distinguished ideals	$2^s \cdot 2^n$

A *tree with level predicates* is a structure of the type $(T; \leq, L_0, \ldots, L_h)$ where $(T; \leq)$ is a tree of height h (where the height of a tree is the maximal number of edges along a maximal path), and for $i \in \{0, \ldots, h\}$, L_i is a unary predicate such that an element $x \in T$ belongs to L_i if and only if x has level i. The next theorem is obtained using a reduction from the EF game problem on embedded equivalence structures.

Theorem 3.6.2. Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins n-round Ehrenfeucht-Fraïssé game $G_n(\mathcal{T}_1, \mathcal{T}_2)$ on finite trees with level predicates $\mathcal{T}_1$ and $\mathcal{T}_2$ of height h. The constant that bounds the running time is $(n+1)^{\cdots^{(n+1)^{(n+1)}}}$ where the tower has height h.

Lastly, we look at *Boolean algebra with distinguished ideals*, which are structures of the form $(A; \leq, 0, 1, I_1, \ldots, I_s)$, where $(A; \leq, 0, 1)$ forms a Boolean algebra and each I_j is an ideal of the algebra $(A; \leq, 0, 1)$. When the domain A is finite, the structure $\mathcal{A}$ can be identified with the structure

$$(2^{X_A}; \subseteq, \emptyset, X_A, 2^{A_1}, \ldots, 2^{A_s}),$$

where each ideal I_i, $1 \leq i \leq s$, is the set 2^{A_i}.

Theorem 3.7.3. Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins the game $G_{n+1}(\mathcal{A}, \mathcal{B})$ on finite Boolean algebras $\mathcal{A} = (2^{X_A}; \subseteq, \emptyset, X_A, 2^{A_1}, \ldots, 2^{A_s})$ and $\mathcal{B} = (2^{X_B}; \subseteq, \emptyset, X_B, 2^{B_1}, \ldots, 2^{B_s})$. The constant that bounds the running time is $2^s \cdot 2^n$.

We summarize all these theorems in Table 1.1. Note that all the time complexity listed are independent on the sizes of the input structures.

The material of this chapter has appeared in Khoussainov/Liu [64, 65].

Chapter 4. The complexity of unary automatic structures

This chapter analyses complexity in *unary automatic structures*. These are infinite structures whose domain is the regular language $1^\star$ and whose relations are recognized by finite automata over the unary alphabet. These structures form an intermediate class between finite structures and automatic structures in general and are interesting due to their proximity to finite structures. One of the advantages possessed by these structures over the automatic structures is the decidability of their monadic second-order theories. Many natural graph problems (such as graph connectivity and reachability) are expressible in monadic second-order logic and are hence decidable for unary automatic graphs. However, deciding these questions by a translation of MSO formulae yields very slow algorithms (super-exponential in the size of the input automatic presentations). In this chapter, we exploit structural properties of unary automatic graphs to solve these questions in polynomial-time. Furthermore, special focus will be put on solving the *isomorphism problem* on a specific subclass $\mathcal{K}$ of unary automatic structures:

INPUT: Given the automatic presentations of two structures $\mathcal{A}$ and $\mathcal{B}$ from $\mathcal{K}$
QUESTION: Decide if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic.

This chapter consists of five sections. The first section introduces unary automatic structures and presents a characterization of these structures. In the second section, we study algorithms on the class of *unary automatic graphs of finite degree*. These are infinite graphs result from a natural unfolding operation applied to finite graphs. In particular, this class of graphs corresponds exactly to the configuration graphs of one-counter processes (pushdown automata with just one stack symbol). Such graphs have received increasing interests in the recent years [33, 107, 112, 32].

We are interested in the following natural decision problem on automatic graphs:

- *Connectivity problem*: Given an automatic graph $\mathcal{G}$, decide whether $\mathcal{G}$ is connected.
- *Reachability problem*: Given an automatic graph $\mathcal{G}$ and two nodes x and y of $\mathcal{G}$, decide whether there is a path from x to y.
- *Infinity testing problem*: Given an automatic graph $\mathcal{G}$ and a node x, decide whether the component in $\mathcal{G}$ containing x is infinite.
- *Infinite component problem*: Given an automatic graph $\mathcal{G}$, decide whether $\mathcal{G}$ has an infinite component.

We present explicit algorithms for all of the problems above. The complexity of each algorithm is polynomial in terms of the sizes of the input automata. For example, we prove the following results.

Table 1.2: Unary automatic graphs of finite degrees

Problems	Complexity
Infinite component problem	$O(n^3)$
Infinite testing problem	$O(n^3)$
Reachability problem	$O(n^4 + \|u\| + \|v\|)$
Connectivity problem	$O(n^3)$
Isomorphism problem	Elementary

Theorem 4.2.11. The infinity testing problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(n^3)$, where n is the size of the input automaton recognizing $\mathcal{G}$. In particular, when $\mathcal{G}$ is fixed, there is a constant time algorithm that decides the infinity testing problem on $\mathcal{G}$.

Theorem 4.2.14. There exists an algorithm that solves the reachability problem on any unary automatic graph $\mathcal{G}$ of finite degree in time $O(p^4 + |u| + |v|)$ where u, v are two input nodes from the graph $\mathcal{G}$ and n is the size of the input automaton recognizing $\mathcal{G}$.

Bouajjani/Esparza/Maler in [9, 24, 111] studied the reachability problem on the class of pushdown graphs which properly contains all unary automatic graphs. They proved that for a pushdown graph and a node v, there is an automaton $\mathcal{A}_v$ that recognizes all nodes reachable from v. This implies decidability of the reachability problem on unary automatic graphs of finite degree. In this work, we provide an alternative algorithm that constructs a deterministic unary automaton $\mathcal{A}_{\mathsf{Reach}}$ that accepts the reachability relation of a unary automatic graph $\mathcal{G}$ of finite degree, hence solving the reachability problem uniformly. This greatly improves the mentioned work of Bouajjani/Esparza/Maler in the class of unary automatic graphs since the automaton constructed now does not depend on the nodes v. The size of the automaton $\mathcal{A}_{\mathsf{Reach}}$ depends only on the size n of the input automaton and the construction takes polynomial time on n.

Corollary 4.2.19. Given a unary automatic graph of finite degree $\mathcal{G}$ represented by an automaton with size n, there is a deterministic automaton $\mathcal{A}_{\mathsf{Reach}}$ with at most $2n^4 + n^3$ states that accepts the reachability relation of $\mathcal{G}$. Furthermore, the time required to construct $\mathcal{A}_{\mathsf{Reach}}$ is $O(n^5)$.

Table 1.2 lists all the problems and their corresponding time complexity.

The rest of this chapter focuses on some natural subclasses of unary automatic structures such as equivalence structures, linear orders and trees and analyses the complexity of deciding the isomorphism problem on these classes of structures.

Table 1.3: The isomorphism problem for classes of unary automatic structures.

Classes of structures	Complexity for deciding the isomorphism problem
Linear orders	$O(n^2)$
Equivalence structures	$O(n)$
Trees	$O(n^4)$

Characterizations of classes of unary automatic structures were given in Blumensath [6] and Khoussainov/Rubin [73]. These results imply that the isomorphism problem for automatic linear orders and equivalence structures are decidable (through monadic second-order interpretations). However, the resulting decision procedures are highly inefficient (doubly- or triply-exponential). In Section 4.3 and Section 4.4, we improve the complexity by providing explicit algorithms in low polynomial time with respect to the input automata.

Theorem 4.3.5. The isomorphism problem for unary automatic linear orders is decidable in quadratic time in the sizes of the input automata.

Theorem 4.4.4. The isomorphism problem for unary automatic equivalence structures is decidable in linear time in the sizes of the input automata.

In Section 4.5, we analyse unary automatic trees. We present a combinatorial characterization for the class of unary automatic trees. This characterization then leads to an algorithm for solving the isomorphism problem.

Theorem 4.5.9. The isomorphism problem for unary automatic trees is decidable in time $O(n^4)$ in the sizes of the input automata.

This chapter also contains an analysis on the *state complexity* of the mentioned classes of unary automatic structures. We define the state complexity of an automatic structure as the smallest number of states needed for automata to describe the domain and relations of the structure. We prove that the state complexity of unary automatic equivalence relations, linear orders and trees are all polynomial with respect to some natural representations of the structures (For each class, we explicitly describe its representation). The study of state complexity of automatic structures is a new, and hopefully fruitful, area.

We obtain the mentioned complexity bounds using detailed analysis on the canonical forms of automatic presentations of structures. The analysis involves lengthy, technical and carefully designed combinatorial arguments. In addition, the analysis greatly interacts with properties of underlying structures. Table 1.3 summarizes the classes of unary automatic structures and their corresponding time complexity for deciding the isomorphism problem.

The material in this chapter has appeared in Khoussainov/Liu/Minnes [66, 67] and Liu/Minnes [86].

Chapter 5. The isomorphism problem for automatic structures

This chapter continues the study of the isomorphism problem for automatic structures in general. Our goal is to investigate the isomorphism problem for some natural classes of automatic structures. Khoussainov/Nies/Rubin/Stephan in [72] has showed that for automatic structures the isomorphism problem is Σ^1_1-complete. The proof exploits the fact that configuration graphs of Turing machines are automatic structures. By direct interpretations, it follows that for the following classes the isomorphism problem is still Σ^1_1-complete [92]: automatic successor trees, automatic undirected graphs, automatic commutative monoids, automatic partial orders, automatic lattices of height 4, and automatic unary functions. On the other hand, the isomorphism problem is decidable for automatic ordinals [77] and automatic Boolean algebras [72]. An intermediate class is the class of locally finite automatic graphs, for which the isomorphism problem is Π^0_3-complete [102].

In this chapter, we solve the following known problems in the area of automatic structures. These problems appear in the list of open problems on automatic structures by Khoussainov/Nerode [71] but have been around for more than 10 years.

(1) Is the isomorphism problem for automatic equivalence structures decidable?

(2) Is the isomorphism problem for automatic linear orders decidable?

(3) Provide natural examples of classes of automatic structures for which the isomorphism is complete for some levels of the arithmetic hierarchy.

(4) Is there always a computable isomorphism between any two isomorphic automatic linear orders (trees)?

We show that for questions (1)(2) and (4) the answer is "no". For question (3) we provide natural classes of automatic structures whose isomorphism problem is Π^0_n-complete for $n \in \mathbb{N}$.

Most of the existing hardness proofs about the isomorphism problem of automatic structures use reductions that involve transition graphs of Turing machines, which are automatic structures. For the class of automatic equivalence structures, linear orders and trees (treated as partial orders), this technique seems to fail for inherent reasons. This is because the relations on these structures are transitive, while the transitive closure of

the configuration graph of a Turing machine is not automatic in general. Hence, new techniques need to be employed. We first prove the following theorem:

Theorem 5.1.5. The isomorphism problem for automatic equivalence relations is Π^0_1-complete.

The proof of this theorem is inspired by the result of Honkala in [56] who shows that it is undecidable whether a rational power series has range $\mathbb{N}$. The proof is a reduction from Hilbert's 10th problem. We follow the ideas Honkala and provide a reduction from Hilbert's 10th problem. The problem consists of deciding if for given two polynomials $p_1(x_1, \ldots, x_k)$ and $p_2(x_1, \ldots, x_k) \in \mathbb{N}[x_1, \ldots, x_k]$ the set $\{(x_1, \ldots, x_k) \in \mathbb{N}^k \mid (\mathbb{N}; +, \times) \models p(x_1, \ldots, x_k) = p_2(x_1, \ldots, x_k)\}$ is non-empty. The celebrated Matiyasevich's theorem proved that the set of pairs of polynomials (p_1, p_2) for which the above set is empty is a Π^0_1-complete set. The crucial part of the reduction involves constructing, for any polynomial $p(x_1, \ldots, x_k) \in \mathbb{N}[x_1, \ldots, x_k]$, an automaton $\mathrm{Run}_{\mathcal{A}[p]}$ over the alphabet $\{a\}^k \times \Sigma$ for some finite alphabet Σ such that for any $x_1, \ldots, x_k \in \mathbb{N}$, $\mathrm{Run}_{\mathcal{A}[p]}$ accepts exactly $p(x_1, \ldots, x_k)$ convoluted words of the form $\otimes(a^{x_1}, \ldots, a^{x_k}, w)$ for some $w \in \Sigma^\star$. In this manner, we encode a polynomial by a regular language. Theorem 5.1.5 then follows from a construction that turns $\mathrm{Run}_{\mathcal{A}[p]}$ into an automatic equivalence structure.

By a direct interpretation, it follows immediately that the isomorphism problem for trees of height 2 is also Π^1_0-complete. The next theorem is proved by induction on n, where the case when $n = 2$ serves as the base case.

Theorem 5.2.13.

1. For any $n \geq 2$, the isomorphism problem for automatic trees of height at most n is Π^0_{2n-3}-complete.

2. The isomorphism problem for the class of automatic trees of finite height is computably equivalent to true arithmetic, i.e., the first-order theory of $(\mathbb{N}; +, \times)$.

Using the same technique and a more elaborate induction, we next prove the following theorem.

Theorem 5.4.10. The isomorphism problem for automatic linear orders is not arithmetic.

A crucial part of the proof of Theorem 5.4.10 is on describing an automatic presentation for the shuffle sum (defined in Chapter 5) of a class of automatic linear orders that are presented in some specific way (see Section 5.4.2.1). Applying Theorem 5.2.13 and Theorem 5.4.10, we obtain information on the Σ^0_k-isomorphisms between automatic structures. The next corollary suggests that, although automatic structures look simple, there may be no "simple" isomorphism between two isomorphism automatic structures.

Corollary 5.5.1. For any $k \in \mathbb{N}$, there exists two isomorphic automatic trees of finite height (and two automatic linear orders) without any Σ^0_k-isomorphisms.

The material in this chapter has appeared in the papers Kuske/Liu/Lohrey [81, 82].

Chapter 6. Computably categorical graphs with finite components

This last chapter focuses on computable structures. In particular, we investigate the computable categoricity of the class of computable strongly locally finite graphs.

Definition 6.1.1. Two computable graphs G_1 and G_2 have the same *computable isomorphism type* if they are computably isomorphic. The number of computable isomorphism types of graph G is called the *computable dimension* of G. If the computable dimension of G equals 1 then G is called *computably categorical*.

It is easy to provide examples of structures whose computable dimension is $\aleph_0$ (e.g. $(\mathbb{N}; \leq)$). The following theorem is due to Goncharov [41].

Theorem 6.1.5. If any two computable presentations of a structure A are Δ^0_2-isomorphic, then the computable dimension of A is either 1 or $\aleph_0$.

In the 1990s and 2000s, Khoussainov/Shore [78], Cholak/Goncharov/Khoussaionov/Shore [12], Hirschfeldt [52] provided examples of structures with various properties whose computable dimensions are natural numbers.

This chapter focuses on the class of computable *strongly locally finite graphs*. They are undirected graphs whose components are all finite. By Goncharov's theorem, it is clear that the computable dimension of any strongly locally finite graph is either 1 or $\aleph_0$. It thus makes perfect sense to work towards a characterization of computably categorical strongly locally finite graphs. This chapter contains a series of results that work towards this characterization.

First, we prove a necessary and sufficient condition for certain types of strongly locally finite graphs to be computably categorical. Let G be a computable strongly locally finite graph. The *size function* $\text{size}_G : \mathbb{N} \to \mathbb{N}$ of a computable graph G maps each node in G (recall that each node in G is itself a number) to the size of the component that contains the node. When size_G is a computable function, we obtain an effective list (without repetition) $C_0, C_1, \ldots$ of all components of G. The *proper extension function* $\text{ext}_G : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ maps any node v in G to the number of components in G that are proper extensions of the component of v.

Theorem 6.3.5. Let G be a computable strongly locally finite graph such that size_G is a computable function. Then the following are equivalent:

1. G is computably categorical.

2. The size function is computable in all computable presentations of G.

3. The function ext_G is computable and there are only finitely many v such that $\text{ext}_G(v) = \infty$.

A *Scott family* for a structure $\mathcal{A}$ is an effective sequence $(\phi_i(\bar{a}, \bar{x}))_{i\in\mathbb{N}}$ of existential formulas, where $\bar{a}$ is a finite sequence of parameters from $\mathcal{A}$, such that the following properties are true:

1. Each formula is satisfiable in $\mathcal{A}$,

2. Each tuple of $\mathcal{A}$ satisfies one of the formulas in the sequence, and

3. Any two tuples that satisfy the same formula can be interchanged by an automorphism of the structure.

It is easy to see that every structure with a Scott family is computably categorical. One can easily show that the theorem above has the following corollary:

Corollary. Let G be a computable strongly locally finite graph such that size_G is a computable function. The graph is computably categorical if and only if it has a Scott family.

Next, we provide a necessary condition for a computable strongly locally finite graph to be computably categorical in the case when the size function is not computable.

Theorem 6.4.1. Let G be a computable strongly locally finite graph. If there exists an infinite Δ^0_2-set of nodes X such that $\text{ext}_G(v) = \infty$ for all $v \in X$, then G is not computably categorical.

The proof of Theorem 6.4.1 uses the priority argument which constructs a computable graph $H \cong G$ that diagonalizes against all computable functions Φ_e by satisfying the following requirement for all $e \in \mathbb{N}$:

$$R_e : \text{ the } e\text{th computable function } \Phi_e \text{ is not an isomorphism from } G \text{ to } H.$$

A natural generalization of the statement in Theorem 6.4.1 is to relax the Δ^0_2 condition for the set X and show that G is not computably categorical whenever there are infinitely many nodes v with $\text{ext}_G(v) = \infty$. However, the next theorem refutes this by constructing a computably categorical strongly locally finite graph that possesses an infinite chain C of embedded components. By the theorem above this set of nodes from the chain C is not a Δ^0_2-set.

Table 1.4: Summary of chapters

<table>
<tr><td colspan="4">Chapter 2: Preliminaries</td></tr>
<tr><td>Chapter 3</td><td colspan="2">Chapter 4,5</td><td>Chapter 6</td></tr>
<tr><td rowspan="3">Finite Structures</td><td colspan="2">Automatic Structures</td><td rowspan="3">Computable Structures</td></tr>
<tr><td>Chapter 4</td><td>Chapter 5</td></tr>
<tr><td>Unary Alphabet</td><td>General Alphabet</td></tr>
</table>

Theorem 6.5.1. There is a strongly locally finite computably categorical graph G that possesses an infinite chain of properly embedded components. In fact, the set of nodes $\{v \mid \text{ext}_G(v) = \infty\}$ is computable in $0''$.

Let G_e be the eth computable graph. Using the tree argument, the proof of Theorem 6.5.1 constructs a computable graph G that satisfies the following requirements for all $e \in \mathbb{N}$:

$$P_e : \text{ if } G_e \cong G \text{ then } G_e \text{ and } G \text{ are computably isomorphic.}$$

we construct G by putting all strategies on the binary tree $2^{<\omega}$. We satisfy all requirements by traversing the tree T along paths of the tree. In the construction, for each graph G_i we select special components A_α in the graph G, where $|\alpha| = i$. The goal is to ensure that along the true path δ the sequence of components $(A_\alpha)_{\alpha \subset \delta}$ forms a chain. The construction will guarantee that the true path can be computed in $0''$.

The proofs of both Theorem 6.3.5 and Theorem 6.4.1 as well as the outline of the proof of Theorem 6.5.1 appeared in Csima/Khoussainov/Liu [14].

As a summary, Table 1.4 illustrates the structures of the topics covered in each chapters of the thesis.

Chapter 2

Preliminaries

We assume basic familiarity with notions and terminologies in model theory, computability theory and automata theory. For completeness of the thesis and to fix notations, some definitions are provided in this chapter. All of the theorems, facts and examples mentioned in this chapter are provided without proofs since the theorems are known among the experts in the area. The references to the proofs are provided in the text. Most of these theorems will be used later.

2.1 Structures

For background on model theory and first-order logic, see standard textbook such as Hodges[54]. We use $\overline{x}$ to denote a tuple $x_1, x_2, \ldots, x_m$ whose length m does not matter. The symbol $\mathbb{N}$ is used for the natural numbers $\{0, 1, 2, \cdots\}$ and $\mathbb{N}_+$ for the positive natural numbers $\{1, 2, \cdots\}$. The symbols $\mathbb{Z}, \mathbb{Q}$ denote respectively the integers and rational numbers.

A *signature* is a finite set τ of relational symbols, where each relational symbol $S \in \tau$ has an associated arity n_S. A *(relational) structure over the signature* τ (or a τ*-structure*) is $\mathcal{A} = (A; (S^{\mathcal{A}})_{s\in\tau})$, where A is a set called the *universe* (or *domain*) of $\mathcal{A}$ and $S^{\mathcal{A}}$ is a relation of arity n_S over the set D, which interprets the relational symbol S. We will assume that every signature contains the equality symbol = and that $=^{\mathcal{A}}$ is the identity relation on $\mathcal{A}$. When the context is clear, we denote $S^{\mathcal{A}}$ with $\mathcal{A}$, and we write $a \in \mathcal{A}$ for $a \in A$.

Note that a signature τ is defined to contain only relational symbols. We consider an m-ary *function* $f : A^m \to A$ as a relation $G(f)$, defined as follows:

$$G(f) = \{(\overline{x}, y) \mid \overline{x} \in A^m, y = f(\overline{x})\}.$$

The relation $G(f)$ is called the *graph of* f. When the context is clear we write f for $G(f)$. We consider constants as 0-ary relations.

A *structure* is a τ-structure for some signature τ. A structure is *finite* if its domain is a finite set; otherwise, the structure is *infinite*. In this thesis, all structures have countable domains.

Two τ-structures $\mathcal{A}$ and $\mathcal{B}$ are *isomorphic*, denoted $\mathcal{A} \cong \mathcal{B}$, if there is a bijection $f : A \to B$ that preserves the relations, i.e.,

$$\forall S \in \tau \forall a_1, a_2, \ldots, a_{n_S} \in A : (a_1, \cdots, a_{n_S}) \in S^{\mathcal{A}} \text{ if and only if } (f(a_1), \cdots, f(a_{n_S})) \in S^{\mathcal{B}}.$$

Here we require that $f(c^{\mathcal{A}}) = c^{\mathcal{B}}$ for all constant symbol $c \in \tau$. In this case, we call the structure $\mathcal{B}$ an *isomorphic copy* of $\mathcal{A}$. The relation of two structures being isomorphic is an equivalence relation and we call the equivalence class of $\mathcal{A}$ the *isomorphism type of* $\mathcal{A}$. In the above definition, the function f is an *isomorphism* from $\mathcal{A}$ to $\mathcal{B}$. For any τ-structure $\mathcal{S}$, a *substructure* of $\mathcal{S}$ is the τ-structure induced on a subset of the universe of $\mathcal{S}$. A *partial isomorphism* from $\mathcal{A}$ to $\mathcal{B}$ is an isomorphism from a substructure of $\mathcal{A}$ to a substructure of $\mathcal{B}$.

Let $\mathcal{A}, \mathcal{B}$ are two structures over the same signature and with disjoint domains. We write $\mathcal{A} \uplus \mathcal{B}$ for the union of the two structures. Hence, when writing $\mathcal{A} \uplus \mathcal{B}$, we implicitly express that the domains of $\mathcal{A}$ and $\mathcal{B}$ are disjoint. More generally, if $\{\mathcal{A}_i \mid i \in I\}$ is a class of pairwise disjoint structures over the same signature, then we denote with $\uplus\{\mathcal{A}_i \mid i \in I\}$ the union of these structures.

The following lists some typical structures and their associated terminologies.

Example 2.1.1 (Structures with unary predicates) *A* structure with unary predicates *has signature* $(P_1, \cdots, P_s)$ *(the value of* s *does not matter) where each* P_i*,* $1 \leq i \leq s$*, is a unary predicate symbol.*

Example 2.1.2 (Graphs) *A* (directed) graph *is considered as a structure* $\mathcal{G} = (V; E)$[1] *where each element in the domain* V *is called a* node *and* $E \subseteq V^2$ *is the* edge relation*. The graph is* undirected *if for all* $u, v \in V$*,* $(u, v) \in E$ *if and only if* $(v, u) \in E$*. The graph* $\mathcal{G}$ *is* of finite degree *if there are at most finitely many edges from each vertex* v*. A* component *of the graph* $\mathcal{G}$ *is the transitive closure of a vertex under the edge relation.*

Example 2.1.3 (Equivalence structures) *An* equivalence structure *is* $\mathcal{E} = (E; \equiv)$ *where* $\equiv \subseteq E^2$ *is an equivalence relation (reflexive, symmetric and transitive). For each element* $e \in E$*, the set* $[e]_{\equiv} = \{x \in E \mid e \equiv x\}$ *is the* $\equiv$-equivalence class *of* e*. The set of equivalence classes partitions the universe* E*. When the context is clear, we simply write* $[e]$ *for* $[e]_{\equiv}$*. By convention, we sometimes use* $(D; E)$ *to denote an equivalence structure with domain* D *and equivalence relation* $E \subseteq D^2$*.*

[1] By convention, we use V instead of G to denote the domain of a graph $\mathcal{G}$.

Example 2.1.4 (Linear orders) *A* linear order *is written as $\mathcal{L} = (L; \leq)$ where $\leq$ is a total partial order. That is, a binary relation on L that is reflexive, anti-symmetric, transitive and for all $x, y \in L$, it is either $(x, y) \in \leq$ or $(y, x) \in \leq$. By convention, we write $x \leq y$ for $(x, y) \in \leq$. Typical examples of infinite linear orders are $(\mathbb{N}; \leq)$, $(\mathbb{Z}; \leq)$ and $(\mathbb{Q}; \leq)$. By convention, we use ω (resp. ζ) to denote the isomorphism type of $(\mathbb{N}; \leq)$ (resp. $(\mathbb{Z}; \leq)$), ω^* to denote the isomorphism type of the negative numbers and* $\mathbf{n}$ *to denote the finite linear order of size n.*

We define the following operations on linear orders. For given linear orders $\mathcal{L}_1 = (L_1; \leq_{\mathcal{L}_1})$ and $\mathcal{L}_2 = (L_2; \leq_{\mathcal{L}_2})$, we denote by $\mathcal{L}_1 + \mathcal{L}_2$ the linear order $(L_1 \times \{1\} \cup L_2 \times \{2\}; \leq)$ where $\leq$ is the relation

$$\Big\{((x_1, 1), (x_2, 1)) \mid x_1, x_2 \in L_1, x_1 \leq_{\mathcal{L}_1} x_2\Big\} \cup \Big\{((y_1, 2), (y_2, 2)) \mid y_1, y_2 \in L_2, y_1 \leq_{\mathcal{L}_2} y_2\Big\} \cup$$
$$\Big\{((x, 1), (y, 2)) \mid x \in L_1, y \in L_2\Big\}.$$

Example 2.1.5 (Trees) *A* tree *is a structure $\mathcal{T} = (T; \leq_T)$, where $\leq_T$ is a partial order on T with a least element, called the* root *of $\mathcal{T}$, and such that for every $x \in T$, the order $\leq_{\mathcal{T}}$ restricted to the set $\{y \mid y \leq_{\mathcal{T}} x\}$ is a finite linear order. We call the relation $\leq_{\mathcal{T}}$ the* ancestry order *or the* tree order *of $\mathcal{T}$ and a node y is an* ancestor *of x (or x is a* descendent *of y) if $y \leq_{\mathcal{T}} x$. The* parent *of x is the immediate ancestor of x (undefined when x is the root) and y is a child of x if x is the parent of y. Elements without children are called* leaves*. Two elements x, y are* incomparable*, denoted by $x|_{\mathcal{T}} y$, if neither $x \leq_{\mathcal{T}} y$ nor $y \leq_{\mathcal{T}} x$.*

The disjoint union of trees form a forest*. We generally use the letter $\mathcal{F}$ to denote a forest and $\leq_{\mathcal{F}}$ to denote the corresponding ancestry order.*

Example 2.1.6 (Boolean algebra) *A* Boolean algebra *is a structure $\mathcal{B} = (B; \leq, 0, 1)$ where $\leq$ is a partial order on B with the maximum element* 1 *and the minimum element* 0 *and satisfies the following properties:*

1. *For all $x, y \in B$, the supremum $\sup\{x, y\}$ and infimum $\inf\{x, y\}$ both exist.*
2. *For all $x \in B$, there is a unique $y \in B$ with $\sup\{x, y\} = 1$ and $\inf\{x, y\} = 0$.*

2.2 Theories

For a signature τ, a *τ-formula* is a formula which uses symbols from τ as non-logical symbols. A *τ-sentence* is a τ-formula without free variables. We use FO to denote the first-order logic. Second-order logic extends FO by including second-order variables that range over relations on the universe, and quantifications over such variables. Monadic second-order logic, denoted by MSO, is the fragment of second-order logic where all second-order variables range over unary relations, i.e., subsets of the universe. By convention, first-order variables are written in small cases: $x, y, z, \ldots$, while monadic second-order variables are

written in upper cases: $X, Y, Z, \ldots$ Without explicitly mention, we write τ-formula (resp. -sentences) for τ-formula (resp. -sentences) in FO.

For a logic L and a τ-structure $\mathcal{A}$, the L-*theory* of $\mathcal{A}$ is the collection of all τ-sentences in L that are satisfied in $\mathcal{A}$. This theory is *decidable* if there is an algorithm that tells whether a given sentence belongs to the theory.

Fix a logic L. Given a τ-structure $\mathcal{A}$, an m-ary relation $R \subset (D^A)^m$ is τ-*definable* in L if there is a τ-formula $\varphi(x_1, \ldots, x_m)$ in L such that

$$\forall x_1, \ldots, x_m \in A : (x_1, \ldots, x_m) \in R \text{ if and only if } \varphi(x_1, \ldots, x_m) \text{ is satisfied in } \mathcal{A}.$$

In this case we say that $\varphi(x_1, \ldots, x_m)$ is an L-*definition* of R. Similarly, a class of τ-structures $\mathcal{K}$ is τ-*definable* in L if there is a τ-sentence φ in L such that $\mathcal{K}$ contains exactly those τ-structures that satisfy φ. For convenience, we will omit the signature τ when the context is clear.

Example 2.2.1 (Binary relations) *Let E be a binary relation symbol. We define the following $\{E\}$-sentence:*

- ref*:* $\forall x : (x, x) \in E$
- sym*:* $\forall x, y : (x, y) \in E \to (y, x) \in E$
- trans*:* $\forall x, y, z : (x, y) \in E \wedge (y, z) \in E \to (x, z) \in E$
- antisym*:* $\forall x, y : (x, y) \in E \wedge (y, x) \in E \to x = y$
- tot*:* $\forall x, y : (x, y) \in E \vee (y, x) \in E$

Hence the class of equivalence structures (resp. linear orders) $(V; E)$ is defined by the first-order sentence ref $\wedge$ sym $\wedge$ trans *(resp.* ref $\wedge$ antisym $\wedge$ trans $\wedge$ tot*).*

Example 2.2.2 (Trees) *The class of trees $(T; \leq_{\mathcal{T}})$ can be defined by the conjunction of* ref $\wedge$ antisym $\wedge$ trans *(treated as a $\{\leq_{\mathcal{T}}\}$-sentence) and the following sentence:*

$$\Big(\forall x, y, z : (y \leq_T x \wedge z \leq_T x) \to (y \leq_T z \vee z \leq_T y)\Big) \wedge \Big(\exists x \forall y : x \leq_T y\Big).$$

One may also view a tree as a graph $(T; E)$, where there is an edge $(u, v) \in E$ if and only if u is the parent of v. It is clear that the edge relation E is $\{\leq_{\mathcal{T}}\}$-definable. Given a tree $\mathcal{T}$, the level *of an element $u \in V$ is the length of the path from the root to u, where the length of a path is the number of E-edges along the path. The* height *of $\mathcal{T}$ is the supremum of the levels of all nodes in V. When the tree $\mathcal{T}$ has height $h \in \mathbb{N}$, the tree order $\leq_{\mathcal{T}}$ is $\{E\}$-definable:*

$$x \leq_{\mathcal{T}} y \Leftrightarrow \bigvee_{0 \leq i \leq h} \Big(\exists x_1 \ldots \exists x_i : x_1 = x \wedge x_i = y \wedge \bigwedge_{0 \leq j < i} (x_j, x_{j+1}) \in E\Big)$$

2.3 The arithmetic hierarchy

For background on Turing machines and computably enumerable sets and degrees, see standard textbooks such as [100, 109]. We use standard Gödel numbering to encode (tuples of) finite objects, e.g., finite sets, finite words, finite structures, automata or machines, etc., into natural numbers. By *computable functions*, we mean partial functions defined on natural numbers that are computable by a Turing machine. It is well-known that there is an effectively list of all computable functions

$$\Phi_0, \Phi_1, \Phi_2, \dots$$

By $\Phi^X_{e,s}(x) = y$, we mean that $e, x, y \leq s$ and the e^{th} computable function, running on input x, with an oracle tape written X outputs y in no more than s steps. We use $\Phi^X_e(x) = y$ to denote that

$$\exists s \in \mathbb{N} : \Phi^X_{e,s}(x) = y.$$

We say that Φ_e *converges on x with oracle X*, denoted by $\Phi^X_e(x)\downarrow$, if $\exists y : \Phi^X_e(x)\downarrow$. Otherwise, Φ_e *diverges on x with oracle X*, and it is denoted by $\Phi^X_e(x)\uparrow$. In the notations above, we omit the oracle symbol X if $X = \emptyset$. For $e \in \mathbb{N}$, let $W_e = \{x \mid \Phi_e(x)\downarrow\}$.

The *characteristic string* of a set $X \subseteq \mathbb{N}$ is an infinite word $w_X \in \{0,1\}^\omega$ such that its ith position $w_X[i] = 1$ if and only if $i \in X$, $i \in \mathbb{N}$. A set $S \subseteq \mathbb{N}$ is *computable in X*, denoted by $S \leq_T X$, if there is $e \in \mathbb{N}$ such that Φ^X_e is a total function and $\Phi^X_e(n) = w_S[n]$ for all $n \in \mathbb{N}$.

Definition 2.3.1 *A set $S \subseteq \mathbb{N}$ is* computably enumerable in X *or* c.e. in X *if there is a computable function Φ_e such that for all $n \in \mathbb{N}$, $n \in S$ if and only if $\Phi^X_e(x)\downarrow$. When $X = \emptyset$, S is* computably enumerable.

A typical example of a set which is computably enumerable but not computable is the *halting problem* $K = \{e \mid \Phi_e(e)\downarrow\}$. It is well-known that a set is computable if and only if both it and its complement are computably enumerable. Therefore the set $\mathbb{N} \setminus K = \{e \mid \Phi_e(e)\uparrow\}$ is not computably enumerable.

In computability theory, the *arithmetic hierarchy* is used to classify subsets of natural numbers with certain first-order definitions.

Definition 2.3.2 *For $n \in \mathbb{N}$, the class Σ^0_n contains all sets A that can be written in the form:*

$$A = \{x \mid (\mathbb{N}; +, \times) \models Q_1 y_1 \cdots Q_n y_n : \varphi(x, y_1, \dots, y_n)\}$$

where $Q_1, Q_2, \cdots$ are the quantifiers $\exists, \forall, \cdots$ and $\varphi(x, y_1, \dots, y_n)$ is a quantifier free formula. The class Π^0_n contains all sets $\mathbb{N} \setminus A$ where $A \in \Sigma^0_n$. The set Δ^0_n is $\Sigma^0_n \cap \Pi^0_n$.

Equivalently, the classes Σ^0_n, Π^0_n, and Δ^0_n can be defined in terms of the relative computability of sets:

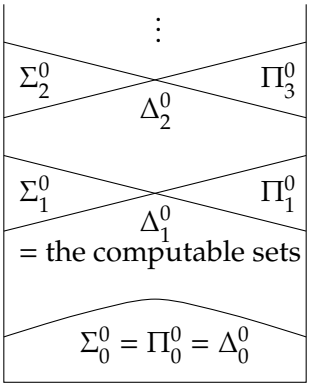


Figure 2.1: The arithmetic hierarchy

- **Base case:** The class Δ_1^0 contains all computable subsets of $\mathbb{N}$. The class Σ_1^0 contains all computably enumerable subsets of $\mathbb{N}$ and the class Π_1^0 contains all subsets of $\mathbb{N}$ whose complements belong to Σ_1^0.
- **Inductive step:** The class Δ_{n+1}^0 contains all subsets of $\mathbb{N}$ computable in some Σ_n^0 sets. The class Σ_{n+1}^0 contains all sets that are computably enumerable in some Σ_n^0 set. The class Π_{n+1}^0 contains all complements of Σ_{n+1}^0 sets.

The sets Σ_n^0 and Π_n^0, $n \in \mathbb{N}$, make up the *arithmetic hierarchy*. See Figure 2.1 for an inclusion diagram (all inclusions are proper). By fixing some effective encoding of strings by natural numbers, we can talk about Σ_n^0-sets and Π_n^0-sets of strings over an arbitrary alphabet. A typical example of a set, which does not belong to the arithmetical hierarchy is *true arithmetic*, i.e., the first-order theory of $(\mathbb{N}; +, \times)$, which we denote by $\mathsf{FOTh}(\mathbb{N}; +, \times)$.

We say that a set $A \subseteq \mathbb{N}$ *m-reduces to* a set $B \subseteq \mathbb{N}$, $A \leq_m B$, if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \mathbb{N}$, $x \in A$ if and only if $f(x) \in B$.

If C is a class of sets, we say that a set $A \subseteq \mathcal{M}$ is *complete for* C (or C*-complete*) if $A \in C$ and for all $B \in C$, $B \leq_m A$. We say that two sets A and B are *computably equivalent* if $A \leq_m B$ and $B \leq_m A$.

Example 2.3.3 (Turing jumps) *For a set $A \subseteq \mathbb{N}$, the* Turing jump *of A is the set $A' = \{x \in \mathbb{N} \mid \Phi_x^A(x) \downarrow\}$. The* nth-jump of A *is defined such that $A^{(0)} = A$ and $A^{(n+1)} = A^{(n)\prime}$. We use $0^{(n)}$ to denote the set $\emptyset^{(n)}$. Note that $0' = K$. It is well-known that for $n \in \mathbb{N}$, $0^{(n)}$ is Σ_n^0-complete.*

Example 2.3.4 (Index sets) *The following index sets are Turing-complete for respective levels of the arithmetic hierarchy (See [109]):*

- *$K_0 = \{e \mid W_e \neq \emptyset\}$ and $K_1 = \{(e, x) \mid x \in W_e\}$ are both Σ_1^0-complete.*
- *EMPTY $= \{e \mid W_e = \emptyset\}$ is Π_1^0-complete.*
- *INF $= \{e \mid W_e$ is infinite $\}$ is Π_2^0-complete.*

– FIN = $\{e \mid W_e$ *is finite* $\}$ *is* Σ_2^0*-complete.*

Example 2.3.5 (Hilbert's 10th problem) *Hilbert's 10th problem asks for deciding if a given Diophantine equation* $p(x_1, \ldots, x_k) = 0$ *has a solution in* $\mathbb{N}_+$ *(for technical reasons, it is useful to exclude* 0 *in solutions). The problem is well-known to be undecidable. The celebrated result of Matiyesevich (See [88]) constructed from a given (index of a) computably enumerable set* $X \subseteq \mathbb{N}$ *a polynomial* $p(x_1, \ldots, x_k) \in \mathbb{Z}[x_1, \ldots, x_k]$ *such that for all* $n \in \mathbb{N}_+$*:* $n \in X$ *if and only if* $\exists y_2, \cdots, y_k \in \mathbb{N}_+ : p(n, y_2, \ldots, y_k) = 0$*. Using a standard encoding of polynomials in* $\mathbb{Z}[x_1, \ldots, x_k]$ *by natural numbers, the following set is Turing complete for* Σ_1^0*:*

$$\{p(x_1, \ldots, x_k) \in Z[x_1, \ldots, x_k] \mid \exists x_1, \cdots, x_k : p(x_1, x_2, \ldots, x_k) = 0\}.$$

2.4 Automata and languages

For backgrounds on automata and language, see standard textbooks such as [57]. In this thesis, by an "automaton", we mean a finite word automaton. Formally, for a fixed alphabet Σ, a *nondeterministic finite automaton (NFA)* is a tuple $\mathcal{A} = (S, \Delta, I, F)$ where S is a set of states, $\Delta \subseteq S \times \Sigma \times S$ is the transition relation, $I \subseteq S$ is a set of initial states, and $F \subseteq S$ is the set of accepting states. We use $\Sigma^\star$ to denote the set of all finite words over alphabet Σ. For $w \in \Sigma^\star$, $|w|$ denotes the length of w.

A *run* of $\mathcal{A}$ on a word $u = a_1 a_2 \cdots a_n \in \Sigma^\star$ is a word of the form

$$r = (q_0, a_1, q_1)(q_1, a_2, q_2) \cdots (q_{n-1}, a_n, q_n) \in \Delta^\star,$$

where $q_0 \in I$. If moreover $q_n \in F$, then r is an *accepting run* of $\mathcal{A}$ on u. We will only use these definitions in case $n > 0$, i.e., we will only speak of nonempty (accepting) runs. The automaton $\mathcal{A}$ is *deterministic* if $|I| = 1$ and for all $q \in S$, $\sigma \in \Sigma$, there is exactly one p with $(q, \sigma, p) \in \Delta$. Hence, a deterministic automaton has precisely one run on each word $r \in \Sigma^\star$. The automaton $\mathcal{A}$ is a *unary automaton* if the alphabet $\Sigma = \{1\}$.

We say the automaton $\mathcal{A}$ *accepts* u if there is an accepting run of $\mathcal{A}$ on u. The *language accepted by* $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the collection of all words over alphabet Σ that are accepted by $\mathcal{A}$. A language is *regular* if it is accepted by some automaton.

The *concatenation* operation on two language L_1, L_2 is defined as $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$. Let ε denote the empty string and $L^0 = \{\varepsilon\}$. For $n \in \mathbb{N}$, let $L^{n+1} = L \cdot L^n$. The *Kleene's star* operation is defined as $L^\star = \cup_{n \in \mathbb{N}} L^n$. The following classical results provide ways to decide if a language is regular.

Kleene's theorem. A language $L \subseteq \Sigma^\star$ is regular if and only if it can be generated from the empty set and singletons by applying a finite number of union, concatenation and the Kleene star operation.

Closure property. The class of regular languages is closed under the set operations, namely, union, intersection and complementation.

Pumping lemma. Suppose $L \subseteq \Sigma^\star$ is a regular language and n is the number of states of an NFA that accepts L. For any word $w \in L$ with $|w| > p$, there are words $x, y, z \in \Sigma^\star$ such that $|y| > 1$, $|xy| \leq n$ and $xy^iz \in L$ for all $i \in \mathbb{N}$.

By the pumping lemma, it is easy to prove that the language $\{0^n1^n \mid n \in \mathbb{N}\}$ is not regular.

Example 2.4.1 (Unary regular languages) *The transition diagram of any automaton over the unary alphabet* $\{1\}$ *is of the following form (See Fig 2.2), where* $i < j$ *are natural numbers. Hence, a language* $U \subseteq \{1\}^\star$ *is regular if and only if there are numbers* $t, \ell \in \mathbb{N}$ *such that* $L = L_1 \cup L_2$ *with* $L_1 \subseteq \{0, \cdots, t-1\}$ *and* L_2 *is a finite union of sets in the form* $\{j + i\ell\}_{i\in\mathbb{N}}$*, where* $t \leq j < t + \ell$*.*

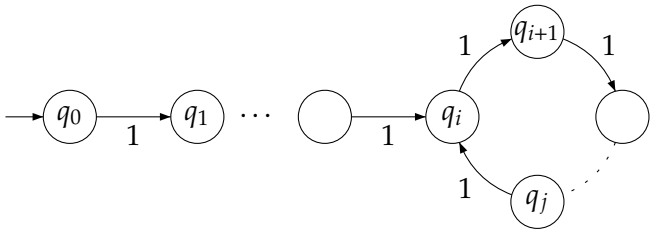


Figure 2.2: NFA over the unary alphabet $\{1\}$.

An NFA $\mathcal{A}$ can be considered as a (finite) representation of the set $L(\mathcal{A}) \subseteq \Sigma^\star$. This notion can be generalized to relations over $\Sigma^\star$ of arbitrary arity n using *synchronous n-tape automata*. Such automata have n input tapes; each of which contains one of the input words. Bits of the n input words are read in parallel until all input strings have been completely processed.

Formally, let $\Sigma_\diamond = \Sigma \cup \{\diamond\}$ where $\diamond$ is a symbol not in Σ. Given n words $w_1, w_2, \ldots, w_n \in \Sigma^\star$, the *convolution* of $(w_1, \ldots, w_n)$ is a word $\otimes(w_1, \ldots, w_n)$ over the alphabet $(\Sigma_\diamond)^n$ with length $\max\{|w_1|, \ldots, |w_n|\}$. The kth symbol of $\otimes(w_1, \ldots, w_n)$ is $(\sigma_1, \ldots, \sigma_n)$ where σ_i is the kth symbol of w_i if $k \leq |w_i|$ and $\diamond$ otherwise. The *relation accepted by a synchronous n-tape automaton* $\mathcal{A}$ is

$$\{(w_1, \ldots, w_n) \mid w_1, \ldots, w_n \in \Sigma^\star, \otimes(w_1, \ldots, w_n) \in L(\mathcal{A})\}.$$

An n-ary relation is *FA-recognizable* or *regular* if it is accepted by some synchronous n-tape automaton. When the context is clear, we refer to a synchronous n-tape automaton, $n \in \{1, 2, \ldots\}$, simply as an NFA. It implies from the closure property that the class of n-ary regular relations is closed under union, intersection and complementation.

For $i \in \{1, \ldots, n\}$, the *projection operation of the ith coordinate* produces from an n-ary relation R an $(n-1)$-ary relation

$$\pi_i(R) = \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \mid \exists x_i : (x_1, \ldots, x_n) \in R\}.$$

From an NFA $\mathcal{A}$ recognizing $R \subseteq (\Sigma^\star)^n$ and $i \in \{1, \ldots, n\}$, one effectively constructs an NFA $\mathcal{A}'$ recognizing $\pi_i(R)$. The automaton $\mathcal{A}'$ can be constructed form $\mathcal{A}$ by omitting the ith tape.

Example 2.4.2 (Linear orderings on words) *Let Σ be a finite alphabet. In the following we describe important examples of regular linear orders over Σ.*

– *We write $\leq_{\text{pref}}$ for the* prefix order *on $\Sigma^\star$, which is defined such that for all $x, y \in \Sigma^\star$, $x \leq_{\text{pref}} y$ if and only if x is a prefix of y. The order $\leq_{\text{pref}}$ is the language*

$$\{\otimes(x, x) \mid x \in \Sigma^\star\} \cdot \{\otimes(\varepsilon, y) \mid y \in \Sigma^\star\}$$

and is recognized by the NFA depicted in Figure.2.3.

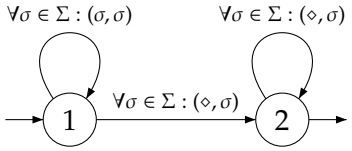


Figure 2.3: The automaton recognizing the prefix order

– *Fix a linear order $<$ on Σ. We write $\leq_{\text{lex}}$ for the* lexicographic order (induced by $<$) *on $\Sigma^\star$, which is defined such that:*

$$x <_{\text{lex}} y \iff x <_{\text{pref}} y \text{ or } \exists z \in \Sigma^\star \exists \sigma, \tau \in \Sigma : x = z\sigma x', y = z\tau y', \sigma < \tau.$$

The order $\leq_{\text{lex}}$ is the language

$$\{\otimes(x, x) \mid x \in \Sigma^\star\} \cdot \{(\sigma, \tau) \in \Sigma^2 \mid \sigma < \tau\} \cdot \{\otimes(y, z) \mid y, z \in \Sigma^\star\}.$$

– *We write $\leq_{\text{llex}}$ for the* length-lexicographic ordering *on $\Sigma^\star$, which is defined as follows:*

$$x <_{\text{llex}} y \iff |x| < |y| \text{ or } (|x| = |y| \wedge x <_{\text{lex}} y).$$

2.5 Automatic structures and computable structures

For detailed background on automatic structures, see the theses [102, 3, 91].

Definition 2.5.1 *Let σ be a signature. An* automatic structure *of signature σ is a σ-structure $\mathcal{A}$ whose domain is a regular language and for each $R \in \sigma$, $R^{\mathcal{A}}$ is FA-recognizable. Any tuple $\mathbb{P}$ of automata that accept the domain and the relations of $\mathcal{A}$ is called an* automatic presentation of $\mathcal{A}$*; in this case, we write $\mathcal{A}(\mathbb{P})$ for $\mathcal{A}$.*

If an automatic structure $\mathcal{A}$ is isomorphic to a structure $\mathcal{A}'$, then $\mathcal{A}$ is called an *automatic copy* of $\mathcal{A}'$ and $\mathcal{A}'$ is *automatically presentable*. In this thesis we sometimes abuse the terminology referring to $\mathcal{A}'$ as simply automatic and calling an automatic presentation of $\mathcal{A}$ also an automatic presentation of $\mathcal{A}'$. We also simplify our statements by saying "given/compute an automatic structure $\mathcal{A}$" for "given/compute an automatic presentation $\mathbb{P}$ of a structure $\mathcal{A}(\mathbb{P})$".

A structure is *unary automatic* if it is automatic and it has an automatic presentation over the unary alphabet $\{1\}$. All finite structures are automatic. The following list important examples of infinite automatic structures.

Example 2.5.2 (A unary automatic structure) *The structure $(\mathbb{N}; \leq)$ is (unary) automatic. An automatic copy of $(\mathbb{N}; \leq)$ is $(1^\star; \{(1^m, 1^n) \mid m \leq n\})$.*

The following proposition from [6] shows that the restriction to a unary alphabet is a natural special case of automatic structures.

Proposition 2.5.3 *Every automatic structure has an automatic copy over the binary alphabet $\{0, 1\}$.*

Example 2.5.4 (Presburger arithmetic) *The structure $(\mathbb{N}; +)$ is automatic but not unary automatic. An automatic copy of the structures is $(\{0, 1\}^\star \cdot 1; +_2)$ where $+_2$ is binary addition when the binary strings are interpreted as the least significant bit first base-2 expansion of the natural numbers. See Figure. 2.4 for the state diagram of an automaton recognizing $+_2$.*

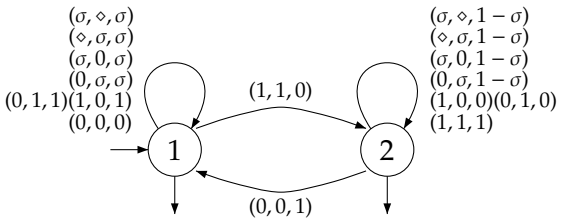


Figure 2.4: The automaton recognizing $+_2$

The following theorem from [102] shows that problems on automatic structures can in fact be reduced to automatic graphs. In this sense, equivalence structure, linear orders, trees can all be considered as special classes of graphs.

Theorem 2.5.5 (Reduction to automatic graphs) *For every structure $\mathcal{A}$ there is a graph $\mathcal{G}(\mathcal{A})$ such that $\mathcal{A}$ is automatic if and only if $\mathcal{G}(\mathcal{A})$ is automatic. Furthermore, an automatic presentation of $\mathcal{G}(\mathcal{A})$ can be constructed in linear time in the size of an automatic presentation of $\mathcal{A}$.*

Example 2.5.6 (Dense linear order) *The dense linear order $(\mathbb{Q}; \leq)$ is automatic. An automatic copy of the structure is $(\{0,1\}^\star \cdot 1; \leq_{\text{lex}})$. To see this, one only needs to prove that the linear order $(\{0,1\}^\star \cdot 1; \leq_{\text{lex}})$ is dense and without endpoints. For denseness, take any $w_1, w_2 \in \{0,1\}^\star \cdot 1$ where $w_1 <_{\text{lex}} w_2$. Then we have two cases:*

- *Case 1: $w_2 = w_1 \cdot x$ for some $x \in \{0,1\}^\star \cdot 1$. Then we have*

$$w_1 <_{\text{lex}} w_1 0^{|x|} 1 <_{\text{lex}} w_2.$$

- *Case 2: $w_1 = x0y$ and $w_2 = x1z$ for some $x, y, z \in \{0,1\}^\star \cdot 1$. Then we have*

$$w_1 <_{\text{lex}} x0y1 <_{\text{lex}} w_2$$

To show that no endpoint exists, take any $w \in \{0,1\}^\star$, and we have

$$w01 <_{\text{lex}} w1 <_{\text{lex}} w11.$$

Therefore

$$(\{0,1\}^\star \cdot 1; \leq_{\text{lex}}) \cong (\mathbb{Q}; \leq).$$

Example 2.5.7 (An automatic equivalence structure) *Let $L \subseteq \Sigma^\star$ be a regular language. Then the structure $(L; \equiv_{\text{len}})$ is automatic, where $x \equiv_{\text{len}} y$ if and only if x and y have the same length.*

Example 2.5.8 *[Configuration graphs of TMs] Let $\mathcal{M}$ be a Turing machine over input alphabet Σ. The* configuration graph *of $\mathcal{M}$ is a graph whose set of nodes consists of all configurations of $\mathcal{M}$, and whose edge relation corresponds to single transitions of $\mathcal{M}$. It is well-known that the configuration graph of any Turing machine is an automatic graph (see Rubin[102]).*

From this fact, it is clear that the reachability problem for automatic graphs is not decidable.

Example 2.5.9 (Non-automaticity) *Examples of structures that are not automatic include:*

- $(\mathbb{N}; \times)$, $(\mathbb{N}; \div)$.
- *The linear order ω^ω.*
- *Atomless Boolean algebra.*
- *The random graph.*

– *The torsion-free Abelian group* $(\mathbb{Q}; +)$ *[114].*

As discussed in Chapter 1, the class of automatic structures form a (proper) subset of the class of *computable structures*.

Definition 2.5.10 *A structure is called* computable *if its domain is a computable subset of natural numbers and all its relations are uniformly computable.*

The definition easily implies that the atomic diagram of a computable structure is computable. On the other hand, almost all other natural properties are undecidable over computable structures. These include reachability, connectedness and even the existence of isolated nodes. Automatic structures possess several nice logical and computability-theoretical properties over the computable structures. Most prominently, the first-order theory of any automatic structure is decidable. The next theorem from [70, 55, 7, 102] singles out this fact as it serves as the main motivation for research in automatic structures. Let $\mathsf{FO} + \exists^{\infty} + \exists^{n,m}$ denote the first-order logic extended by the quantifier $\exists^{\infty}$ (there exist infinitely many) and $\exists^{n,m}$ (there exist finitely many and the exact number is congruent to n modulo m, where $m, n \in \mathbb{N}$).

Theorem 2.5.11 *From an automatic presentation* $\mathbb{P}$ *and a formula* $\varphi(\overline{x}) \in \mathsf{FO} + \exists^{\infty} + \exists^{n,m}$ *in the signature of* $\mathcal{A}(\mathbb{P})$*, one can compute an automaton whose language consists of those tuples* $\overline{a}$ *from* $\mathcal{A}(\mathbb{P})$ *that make* φ *true. In particular, the* $\mathsf{FO} + \exists^{\infty} + \exists^{n,m}$ *theory of any automatic structure* A *is (uniformly) decidable.*

Chapter 3

The Complexity of Ehrenfeucht-Fraïssé Games

In finite model theory, Ehrenfeucht-Fraïssé game is an important tool in illustrating the expressive power of first-order logic. In particular, for two structures $\mathcal{A}$ and $\mathcal{B}$ with the same signature, Duplicator wins the n-round Ehrenfeucht-Fraïssé game on $\mathcal{A}$ and $\mathcal{B}$ if and only if $\mathcal{A}$ and $\mathcal{B}$ agree on all FO sentences of quantifier rank up to n. Hence, Ehrenfeucht-Fraïssé games reveal information on the degree of "similarity" between structures. We concern the following problem: Given $n \in \omega$ as a parameter, and two relational structures $\mathcal{A}$ and $\mathcal{B}$ with the same signature, who is the winner of the n-round EF game played on $\mathcal{A}$ and $\mathcal{B}$? In this chapter, we focus on the efficiency of answering the above question for standard classes of structures such as trees, Boolean algebras, equivalence structures and some of their expansions. All structures we consider are finite. For each of the studied classes, we provide an algorithm that decides the winner of an n-round EF games played on structures in the class. Assuming n is a constant, all algorithms run in constant time. The values of the constants are bounded by functions on n. Clearly, the constants obtained depend on the representations of the structures. In each case, it will be clear from the content how we represent our structures.

3.1 Ehrenfeucht-Fraïssé games

Ehrenfeucht-Fraïssé (EF) games constitute an important tool in both finite and infinite model theory. For example, in infinite model theory these games can be used to prove the Scott isomorphism theorem which states that all countable structures are described (up to isomorphism) in the infinitary logic $L_{\omega_1,\omega}$ [106]. In finite model theory, these games and their different versions are used to establish expressibility results in first-order logic and its extensions [63]. The reader can find these results in standard books on finite and infinite

model theory (e.g. [54, 85]) and in relatively recent papers (e.g. [15]).

Definition 3.1.1 *Let $\mathcal{A}$ and $\mathcal{B}$ be structures and $n \in \mathbb{N}$. We define the n-*round EF game *on $\mathcal{A}$ and $\mathcal{B}$, denoted by $G_n(\mathcal{A}, \mathcal{B})$, as follows. There are two players, Duplicator and Spoiler, both are provided with $\mathcal{A}$ and $\mathcal{B}$. The game consists of n rounds. Informally, Duplicator's goal is to show that these two structures are similar, while Spoiler needs to show the opposite. At round i, Spoiler selects structure $\mathcal{A}$ or $\mathcal{B}$, and then takes an element from the selected structure. Duplicator responds by selecting an element from the other structure.*

*A k-*round play*, $k \leq n$, produced by the players is a sequence of elements $(a_1, b_1), \cdots, (a_k, b_k)$, where $a_i \in A$ and $b_i \in B$ for $i = 1, \ldots, k$; and if Spoiler selects a_i (or b_i) then Duplicator select b_i (or a_i). Duplicator* wins an n-round play *if the mapping $a_i \to b_i$, $i = 1, \ldots, n$, extended by mapping the element $c^{\mathcal{A}}$ to $c^{\mathcal{B}}$ where c is a constant symbol in the signature of $\mathcal{A}$ and $\mathcal{B}$, is a partial isomorphism between $\mathcal{A}$ and $\mathcal{B}$.*

We are concerned with the *n-Ehrenfeucht-Fraïssé problem*, where $n \in \mathbb{N}$, defined as follows:

INPUT: Two structures $\mathcal{A}$ and $\mathcal{B}$ with the same signature
QUESTION: Does Duplicator win the game $G_n(\mathcal{A}, \mathcal{B})$?

It is clear that if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic then Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ regardless of the value of n. The opposite is not always true. The *quantifier rank* of a τ-formula φ, $\mathrm{qr}(\varphi)$, measures the depth of quantifier nesting in φ and is defined as follows:

$$\mathrm{qr}(\varphi) = \begin{cases} 0 & \text{if } \varphi \text{ is atomic} \\ \max\{\mathrm{qr}(\varphi_1), \mathrm{qr}(\varphi_2)\} & \text{if } \varphi \text{ is } \varphi_1 \mathsf{OP} \varphi_2, \mathsf{OP} \in \{\vee, \wedge\} \\ \mathrm{qr}(\varphi_0) & \text{if } \varphi \text{ is } \neg\varphi_0 \\ \mathrm{qr}(\varphi_0) + 1 & \text{if } \varphi \text{ is } \mathsf{Q}x : \varphi_0(x), \mathsf{Q} \in \{\exists, \forall\} \end{cases}$$

We use $\mathsf{FO}[n]$ to denote the set of all first-order sentences of quantifier rank up to n. The following fundamental theorem provides the main motivation for studies on EF games.

Theorem 3.1.2 (Ehrenfeucht-Fraïssé) *For $n \in \mathbb{N}$, Duplicator wins the n-round EF game $G_n(\mathcal{A}, \mathcal{B})$ on two structures $\mathcal{A}$ and $\mathcal{B}$ in the same signature if and only if $\mathcal{A}$ and $\mathcal{B}$ agree on $\mathsf{FO}[n]$.*

Let $\mathcal{A}$ and $\mathcal{B}$ be two finite structures in the same signature and $n = \min\{|A|, |B|\}$. Since two finite structures are elementary equivalent if and only if they are isomorphic (see for example [54]), by Theorem 3.1.2, Duplicator wins the EF game $G_n(\mathcal{A}, \mathcal{B})$ if and only if $\mathcal{A} \cong \mathcal{B}$. Thus, we can consider solving the n-EF problem as an approximation to the isomorphism problem.

One can do the following rough estimates for finding the winner of the game $G_n(\mathcal{A}, \mathcal{B})$. There are finitely many, up to logical equivalence, formulas $\varphi_1, \ldots, \varphi_k$ of quantifier rank

n (see for example [85]). By Theorem 3.1.2, answering the Ehrenfeucht-Fraïssé problem on $\mathcal{A}$ and $\mathcal{B}$ reduces to checking whether for all $i \in \{1, \ldots, k\}$, the structure $\mathcal{A}$ satisfies φ_i if and only if $\mathcal{B}$ satisfies φ_i. Since $\mathcal{A}$ and $\mathcal{B}$ are finite, this problem can be solved in polynomial time in terms of the sizes of $\mathcal{A}$ and $\mathcal{B}$. However, there are two important issues here. The first issue concerns the number k that depends on n; k is approximately bounded by the n-repeated exponentiation of 2. The second issue concerns the degree of the polynomial for the running time that is bounded by n. Thus, questions arise as to for which standard structures the value of k is feasible as a function of n, and whether the degree of the polynomial for the running time can be made small.

As an example, for the class of finite linear orders, the following theorem is well-known (see [42]).

Theorem 3.1.3 *For any $n \in \mathbb{N}$ and finite linear orders $\mathcal{L}_1, \mathcal{L}_2$, Dupicator wins the EF game $G_n(\mathcal{L}_1, \mathcal{L}_2)$ if and only if $|L_1| = |L_2|$ or $|L_1| \geq 2^n - 1$ and $|L_2| \geq 2^n - 1$.*

The above theorem suggests an algorithm such that, assuming the lengths of the input finite linear orders are given explicitly in their representations, the n-Ehrenfeucht-Fraïssé problem on the class of finite linear orders can be answered in constant time. Therefore in this example, the number k roughly equals to 2^n, and the degree of the polynomial for the running time is 0.

In the subsequent sections, we exploit structural properties in structures with unary predicates, equivalence structures, trees and Boolean algebra to obtain algorithms for solving the n-Ehrenfeucht-Fraïssé problem.

3.2 Simple example: structures with unary predicates

This is an elementary section that gives a full solution for EF games in the case when the language contains unary predicates only. Here is the main result of this section.

Theorem 3.2.1 *Fix the signature $\sigma = \{P_1, \ldots, P_s\}$, where each P_i is a unary predicate symbol. Let $n \in \omega$. There exists an algorithm that runs in constant time and decides whether Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ on finite σ-structures $\mathcal{A}$ and $\mathcal{B}$. The constant that bounds the running time is $2^s \cdot n$.*

Let $\sigma = \{P_1, \ldots, P_s\}$ be a collection of unary predicate symbols and $\mathcal{A} = (A; P_1, P_2, \cdots, P_s)$, $\mathcal{B} = (B; P_1, P_2, \cdots, P_s)$ be two σ-structures. For any σ-structure C, set $P_{s+1}^C = \bigcap_i C \setminus P_i^C$. We prove Theorem 3.2.1 using the following two lemmas.

Lemma 3.2.2 *Suppose $P_1, P_2, \cdots, P_s$ are pairwise disjoint. Then Duplicator wins $G_n(\mathcal{A}, \mathcal{B})$ if and only if for all $i \in \{1, \ldots, s+1\}$ if $|P_i^{\mathcal{A}}| < n$ or $|P_i^{\mathcal{B}}| < n$ then $|P_i^{\mathcal{A}}| = |P_i^{\mathcal{B}}|$. In particular, when Duplicator wins, it is the case that for all $i \in \{1, \ldots, s+1\}$, $|P_i^{\mathcal{A}}| > n$ if and only if $|P_i^{\mathcal{B}}| > n$.*

Proof. To prove the lemma, suppose that there is $i \in \{1, \ldots, k+1\}$ such that $|P_i^{\mathcal{A}}| < n$ but $|P_i^{\mathcal{A}}| \neq |P_i^{\mathcal{B}}|$. Assume $|P_i^{\mathcal{B}}| < |P_i^{\mathcal{A}}|$. Then Spoiler selects $|P_i^{\mathcal{A}}|$ elements from $P_i^{\mathcal{A}}$. This strategy is clearly winning for Spoiler as in the first $|P_i^{\mathcal{B}}|$ rounds, Duplicator has to respond by choosing distinct elements from $P_i^{\mathcal{B}}$ and in the $|P_i^{\mathcal{B}}| + 1^{\text{th}}$ round, an element not in $P_i^{\mathcal{B}}$ has to be chosen and the partial isomorphism cannot be maintained. Similarly, if $|P_i^{\mathcal{B}}| > |P_i^{\mathcal{A}}|$, Spoiler wins by selecting $\min\{n, |P_i^{\mathcal{B}}|\}$ elements from $P_i^{\mathcal{B}}$.

Conversely, assume that for all $i \in \{1, \ldots, s+1\}$, it is either that $|P_i^{\mathcal{A}}| = |P_i^{\mathcal{B}}|$ or $|P_i^{\mathcal{A}}|$ and $|P_i^{\mathcal{B}}|$ are both greater than n. Duplicator has a winning strategy as follows: At round k, assume that the players have produced the k-round play $(a_1, b_1), \cdots, (a_k, b_k)$. If Spoiler selects $a_{k+1} \in A$, then Duplicator responds by selecting $b_{k+1} \in B$ where

- If $a_{k+1} = a_i$ for some $i \in \{1, \ldots, k\}$ then $b_{k+1} = b_i$.
- Otherwise, let $j \in \{1, \ldots, s+1\}$ be such that $a_{k+1} \in P_j^{\mathcal{A}}$, there must be some $b \in P_j^{\mathcal{B}}$ such that $b \notin \{b_1, \ldots, b_k\}$. Let $b_{k+1} = b$.

The cases when Spoiler selects an element from B are treated similarly. The strategy is clearly winning. ∎

Now assume that for a structure $\mathcal{A}$, the unary predicates $P_1, P_2, \ldots, P_s$ are not necessarily pairwise disjoint. For each element $x \in A$, define the *characteristic* of x, $\text{ch}(x)$, as a binary word $t_1 t_2 \cdots t_s \in \{0,1\}^s$ such that for each $1 \leq i \leq s$, $t_i = 1$ if $x \in P_i$ and $t_i = 0$ otherwise. There are 2^s pairwise distinct characteristics, and we order them in lexicographic order: $c_1, \ldots, c_{2^s}$. Construct the structure $\mathcal{A}' = (A; Q_1, \cdots, Q_{2^s})$ such that for all $1 \leq i \leq 2^s$, $Q_i = \{x \in A \mid \text{ch}(x) = c_i\}$. The following is now an easy lemma.

Lemma 3.2.3 *Duplicator wins $G_n(\mathcal{A}, \mathcal{B})$ if and only if Duplicator wins $G_n(\mathcal{A}', \mathcal{B}')$.*

Proof of Theorem 3.2.1. The above lemmas give us the following algorithm for solving the game $G_n(\mathcal{A}, \mathcal{B})$: We represent each of $\mathcal{A}$ and $\mathcal{B}$ by 2^s lists, and the ith list lists all elements with characteristic c_i. By Lemma 3.2.3, to solve the game $G_n(\mathcal{A}, \mathcal{B})$, it is sufficient to solve the game $G_n(\mathcal{A}', \mathcal{B}')$. The algorithm then checks the conditions in Lemma 3.2.2 by reading the lists. In each list it reads at most n elements. Hence, the process takes time bounded by $2^s \cdot n$. ∎

3.3 Equivalence structures

In this section we study EF games played on equivalence structures. For a finite equivalence structure $\mathcal{E}$, we list all the equivalence classes of $\mathcal{E}$ as $E_1, \ldots, E_k$ such that $|E_i| \leq |E_{i+1}|$ for all $1 \leq i < k$. Let $q_{\mathcal{E}}$ be the number of equivalence classes in $\mathcal{E}$; for each $t < n$, let $q_t^{\mathcal{E}}$ be the number of equivalence classes in $\mathcal{E}$ with size t. Finally, let $q_{\geq r}^{\mathcal{E}}$ be the number of equivalence classes in $\mathcal{E}$ of size at least r. Let $\mathcal{A}$ and $\mathcal{B}$ be two finite equivalence structures.

Lemma 3.3.1 *If Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ on equivalence structures $\mathcal{A}$ and $\mathcal{B}$, then the following must be true:*

(1) If $q_{\mathcal{A}} < n$ or $q_{\mathcal{B}} < n$ then $q_{\mathcal{A}} = q_{\mathcal{B}}$; and

(2) $q_{\mathcal{A}} \geq n$ if and only if $q_{\mathcal{B}} \geq n$.

Proof. To prove the lemma, we assume that one of the two statements (1) or (2) is false. Suppose (1) is false and Say $q_{\mathcal{A}} < n$. If $q_{\mathcal{A}} < q_{\mathcal{B}}$ ($q_{\mathcal{B}} < q_{\mathcal{A}}$), Spoiler wins the game $G_n(\mathcal{A}, \mathcal{B})$ by first selecting $q_{\mathcal{A}}$ ($q_{\mathcal{B}}$) pairwise-nonequivalent elements from $\mathcal{A}$ ($\mathcal{B}$) and then selecting an element in $\mathcal{B}$ ($\mathcal{A}$) that is not equivalent to any elements selected by Duplicator. Suppose (2) is false. Say $q_{\mathcal{A}} \geq n$ but $q_{\mathcal{B}} < n$. Spoiler wins the game by selecting n elements from distinct equivalence classes in $\mathcal{A}$. Hence the lemma is proved. ∎

By Lemma 3.3.1, in our analysis below, we always assume that $q_{\mathcal{A}} = q_{\mathcal{B}}$ or $q_{\mathcal{A}} \geq n$ if and only if $q_{\mathcal{B}} \geq n$. We need the following notation for the next lemma and definition. For $t \leq n$, let $q^t = \min\{q^{\mathcal{A}}_{\geq t}, q^{\mathcal{B}}_{\geq t}\}$. Let $\mathcal{A}_t$ and $\mathcal{B}_t$ be equivalence structures obtained by taking out exactly q^t equivalence classes of size $\geq t$ from $\mathcal{A}$ and $\mathcal{B}$ respectively. We also set $n - q^t$ to be 0 in case $q^t \geq n$; and otherwise $n - q^t$ has its natural meaning.

Lemma 3.3.2 *Spoiler wins the game $G_n(\mathcal{A}, \mathcal{B})$ when any one of the following holds:*

1. *There is some $t < n$ such that $q^{\mathcal{A}}_t \neq q^{\mathcal{B}}_t$ and $n - \min\{q^{\mathcal{A}}_t, q^{\mathcal{B}}_t\} > t$.*
2. *There is some $t \leq n$ such that $n - q^t > 0$ and one of the structures $\mathcal{A}_t$ or $\mathcal{B}_t$ has an equivalence class of size $\geq n - q^t$ and the other structure does not.*

Proof. To prove the first part of the lemma, assume that $q^{\mathcal{A}}_t > q^{\mathcal{B}}_t$ and $n - q^{\mathcal{B}}_t > t$. Spoiler's strategy is the following: First, select elements $a_1, \ldots, a_{q^{\mathcal{B}}_t}$ from distinct equivalence classes of size t in $\mathcal{A}$. Duplicator must select elements $b_1, \ldots b_{q^{\mathcal{B}}_t}$ also from distinct equivalence classes of size t in $\mathcal{B}$ as otherwise, Duplicator will clearly lose. Next, Spoiler selects t distinct elements $x_1, \ldots, x_t$ in the equivalence class of size t in $\mathcal{A}$. If Duplicator responds by choosing elements $y_1, \ldots, y_t$ in an equivalence class of size $< t$ then Duplicator would clearly lose. Hence Duplicator must select all $y_1, \ldots, y_t$ from an equivalence class Y of size $> t$. After t moves, Spoiler selects a new element in Y, thus winning the game. The case when $q^{\mathcal{B}}_t > q^{\mathcal{A}}_t$ and $n - q^{\mathcal{A}}_t > t$ is proved similarly.

For the second part, assume $\mathcal{A}_t$ has an equivalence class of size $\geq n - q^t$ and $\mathcal{B}_t$ does not, Spoiler has the following winning strategy. Spoiler selects q^t pairwise non-equivalent elements $a_1, \ldots, a_{q^t}$ in $\mathcal{A}$ from equivalence classes of size greater than or equal to t. Let $b_1, \ldots, b_{q^t}$ be elements selected by Duplicator. Note that for each i, the size of the equivalence class $[b_i]$ is greater than or equal to t. Otherwise, if the size of $[b_i]$ were smaller than t, then the size of $[b_i]$ would be smaller than $n - q^t$. Hence, in this case, Spoiler would win by

selecting elements from $[a_i]$. Now let X be an equivalence class of size $\geq n - q^t$ as stipulated in the lemma. Spoiler wins the game by selecting $n - q^t$ distinct elements in X. ∎

We now single out the hypothesis of the lemma above and give the following definition.

Definition 3.3.3 1. *We say that $G_n(\mathcal{A}, \mathcal{B})$ has* small disparity *if there is some $t < n$ such that $q_t^{\mathcal{A}} \neq q_t^{\mathcal{B}}$ and $n - \min\{q_t^{\mathcal{A}}, q_t^{\mathcal{B}}\} > t$.*

2. *We say that $G_n(\mathcal{A}, \mathcal{B})$ has* large disparity *if there is some $t \leq n$ such that $n - q^t > 0$ and one of the structures $\mathcal{A}_t$ or $\mathcal{B}_t$ has an equivalence class of size $\geq n - q^t$ and the other structure does not.*

Lemma 3.3.4 *Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ if and only if $G_n(\mathcal{A}, \mathcal{B})$ has neither small nor large disparity.*

Proof. The previous lemma proves one direction. For the other, we assume that neither small nor large disparity occurs in the game. We describe a winning strategy for Duplicator.

Let us assume that the players have produced a k-round play $(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)$. In case $k = 0$, we are at the start of the game $G_n(\mathcal{A}, \mathcal{B})$. Our inductive assumptions on this k-round play are the following:

(1) For all $i, j \in \{1, \ldots, k\}$, $a_i \equiv^{\mathcal{A}} a_j$ if and only if $b_i \equiv^{\mathcal{B}} b_j$, and the map $a_i \to b_i$ is injective.

(2) For all $i \in \{1, \ldots, k\}$, $|[a_i]| \geq n - i$ if and only if $|[b_i]| \geq n - i$.

(3) For all $i \in \{1, \ldots, k\}$, $|[a_i]| < n - i$ then $|[a_i]| = |[b_i]|$.

(4) Let $\mathcal{A}'$ and $\mathcal{B}'$ be the equivalence structures obtained by removing the equivalence classes $[a_1], \ldots, [a_k]$ from $\mathcal{A}$ and the equivalence classes $[b_i], \ldots, [b_k]$ from $\mathcal{B}$, respectively. We assume that $\mathcal{A}'$ and $\mathcal{B}'$ satisfy the following conditions:

   (a) In game $G_{n-k}(\mathcal{A}', \mathcal{B}')$ no small disparity occurs.

   (b) In game $G_{n-k}(\mathcal{A}', \mathcal{B}')$ no large disparity occurs.

Assume that Spoiler selects an element $a_{k+1} \in A$. Duplicator responds to this move by choosing b_{k+1} as follows: If $a_{k+1} = a_i$ then $b_{k+1} = b_i$. Otherwise, if $E(a_i, a_{k+1})$ is true in $\mathcal{A}$ then Duplicator chooses a new b_{k+1} such that $E(b_i, b_{k+1})$ is true in $\mathcal{B}$. Assume a_{k+1} is not equivalent to any of the elements $a_1, \ldots, a_k$. If $|[a_{k+1}]| \geq n-k$ then Duplicator chooses any b_{k+1} that is not equivalent to any of the elements $b_1, \ldots, b_k$ and $|[b_{k+1}]| \geq n - k$. Duplicator can select such an element as otherwise large disparity would occur in the game. If $|[a_{k+1}]| < n - k$ then Duplicator chooses b_{k+1} such that $|[b_{k+1}]| = |[a_{k+1}]|$ and b_{k+1} is not equivalent to any of the elements $b_1, \ldots, b_k$. There exists such an element b_{k+1} for Duplicator to choose as otherwise

small disparity would occur in the game. The case when Spoiler selects an element from B is treated similarly.

We now show that the $(k+1)$-round play $(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k), (a_{k+1}, b_{k+1})$ satisfies the inductive assumptions. Inductive assumptions (1), (2), and (3) can easily be checked to be preserved. To show that assumption (4) is preserved, consider the equivalence structures $\mathcal{A}''$ and $\mathcal{B}''$ obtained by removing the equivalence classes $[a_1]$, ..., $[a_k]$, $[a_{k+1}]$ from $\mathcal{A}$ and the equivalence classes $[b_1]$, ..., $[b_k]$, $[b_{k+1}]$ from $\mathcal{B}$, respectively. In game $G_{n-k-1}(\mathcal{A}'', \mathcal{B}'')$ small disparity does not occur as otherwise game $G_{n-k}(\mathcal{A}', \mathcal{B}')$ would have small disparity. Thus, assumption (4*a*) is also preserved. Similarly, if $G_{n-k-1}(\mathcal{A}'', \mathcal{B}'')$ had large disparity then game $G_{n-k}(\mathcal{A}', \mathcal{B}')$ would also have large disparity contradicting the inductive assumption. Hence the strategy described must be a winning strategy due to the fact that Duplicator preserves the inductive assumption (1) at each round. ∎

Theorem 3.3.5 *Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ on finite equivalence structures $\mathcal{A}$ and $\mathcal{B}$. The constant that bounds the running time is n.*

Proof. This result follows from Lemma 3.3.4. We represent each equivalence structure $\mathcal{A}$ and $\mathcal{B}$ in two lists. For example, the first list for the structure $\mathcal{A}$ lists all equivalence classes of $\mathcal{A}$ in increasing order; the second list is $q^{\mathcal{A}}$, $q_1^{\mathcal{A}}$, $q_{\geq 1}^{\mathcal{A}}$, $q_2^{\mathcal{A}}$, $q_{\geq 2}^{\mathcal{A}}$, ... The algorithm runs through the second lists for both $\mathcal{A}$ and $\mathcal{B}$, and for each $t \leq n$ checks whether or not small or large disparity occurs. If the algorithm detects disparity then Spoiler wins, otherwise, Duplicator wins. ∎

This theorem can be extended to equivalence structures expanded with unary predicates that act on equivalence classes uniformly as explained in the following definition.

Definition 3.3.6 *A* homogeneously colored equivalence structure *is* $(A; \equiv, P_1, \ldots, P_s)$ *where*

- $(A; \equiv)$ *is an equivalence structure; and*
- *Each P_i is a homogeneous unary relation on A meaning that for all $x, y \in A$ if $x \equiv y$ then $x \in P_i$ if and only if $y \in P_i$.*

Let $\mathcal{A} = (A; E, P_1, \ldots, P_s)$ be a homogeneously colored equivalence structure. As in the previous section, we define the characteristic $\mathrm{ch}(x)$ of an element $x \in A$ as a binary word $t_1 t_2 \ldots t_s \in \{0,1\}^s$ such that for each $1 \leq i \leq s$, $t_i = 1$ if $x \in P_i$ and $t_i = 0$ otherwise. Since each predicate P_i is homogeneous, any pair of equivalent elements of $\mathcal{A}$ have the same characteristics. We put all 2^s characteristics in a list $c_1, c_2, \ldots, c_{2^s}$. Therefore we can represent $\mathcal{A}$ as a disjoint union of equivalence structures $\mathcal{A}_1, \ldots, \mathcal{A}_{2^s}$, where A_ε is the subset of A consisting of elements with characteristic c_ε. The above theorem can thus be extended to the following result:

Theorem 3.3.7 *There exists an algorithm that runs in constant time and decides whether Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ on finite homogeneously colored equivalence structures $\mathcal{A}$ and $\mathcal{B}$. The constant that bounds the running time is $2^s \cdot n$.* ∎

3.4 Equivalence structures with colors

In this section we study structures $\mathcal{A}$ of the form $(A; E, P_1, \ldots, P_s)$, where E is an equivalence relation on A and $P_1, \ldots, P_s$ are unary predicates on A. Note that $P_1, ..., P_s$ are not necessarily homogeneous unary predicates. We call these structures *equivalence structures with s colors*. Our goal is to give a full solution for EF games played on equivalence structures with s colors. We start with the case when $s = 1$. The case for $s > 2$ will be explained later.

Let $\mathcal{A} = (A; E, P)$ be a finite equivalence structure with one color. We say $x \in A$ is *colored* if $P(x)$ is true; otherwise x is *uncolored*. We say that an equivalence class X has *type* $\text{tp}(X) = (i, j) \in \mathbb{N}^2$, if the number of colored elements of X is i, non-colored elements is j; thus $i + j = |X|$.

Definition 3.4.1 *Given two types (i, j) and (i', j') respectively. We say that (i, j) is* colored n-equivalent *to (i', j'), denoted by $(i, j) \equiv_n^C (i', j')$, if the following holds.*

1. *If $i < n$ then $i' = i$, otherwise $i' \geq n$.*
2. *If $j < n - 1$ then $j' = j$, otherwise $j' \geq n - 1$.*

We say that (i, j) is non-colored n-equivalent *to (i', j'), denoted by $(i, j) \equiv_n^N (i', j')$, if the following holds.*

1. *If $j < n$ then $j' = j$, otherwise $j' \geq n$;*
2. *If $i < n - 1$ then $i' = i$, otherwise $i' \geq n - 1$.*

For $X \subseteq A$, we use $(X; E \restriction X, P \restriction X)$ to denote the equivalence structure obtained by restricting E and P on X. Note that given two equivalence classes X and Y of types (i, j) and (i', j') respectively, if (i, j) is colored (non-colored) n-equivalent to (i', j'), then Duplicator wins the n-round game played on structures $(X; E \restriction X, P \restriction X)$ and $(Y, E \restriction Y, P \restriction Y)$, given the fact that Spoiler chooses a colored (non-colored) element in the first round. The following lemma follows from the definition above.

Lemma 3.4.2 *If $(i', j') \equiv_n^C (i, j)$ or $(i', j') \equiv_n^N (i, j)$, then $(i', j') \equiv_{n-1}^C (i, j)$ and $(i', j') \equiv_{n-1}^N (i, j)$.*

For a finite equivalence structure $\mathcal{A} = (A; E, P)$ with one color, we need the following notations:

– For type (i, j) and $k \geq 1$, set $C^{\mathcal{A}}_{(i,j),k}$ as the set

$$\{X \mid X \text{ is an equivalence class of } \mathcal{A} \text{ and } \mathrm{tp}(X) \equiv^{C}_{k} (i, j)\}.$$

Set $N^{\mathcal{A}}_{(i,j),k}$ as the set

$$\{X \mid X \text{ is an equivalence class of } \mathcal{A} \text{ and } \mathrm{tp}(X) \equiv^{N}_{k} (i, j)\}.$$

– For type (i, j) and $k \geq 1$, set

$$q^{\mathcal{A},C}_{(i,j),k} = |C^{\mathcal{A}}_{(i,j),k}| \text{ and } q^{\mathcal{A},N}_{(i,j),k} = |N^{\mathcal{A}}_{(i,j),k}|.$$

– For $\mathcal{A}$ and $\mathcal{B}$, set

$$q^{C}_{(i,j),k} = \min\{q^{\mathcal{A},C}_{(i,j),k}, q^{\mathcal{B},C}_{(i,j),k}\} \text{ and } q^{N}_{(i,j),k} = min\{q^{\mathcal{A},N}_{(i,j),k}, q^{\mathcal{B},N}_{(i,j),k}\}.$$

– Set $\mathcal{A}^{C}((i, j), k)$ as the structure obtained from $\mathcal{A}$ by removing $q^{C}_{(i,j),k}$ equivalence classes in $C^{\mathcal{A}}_{(i,j),k}$.

– Set $\mathcal{A}^{N}((i, j), k)$ as the structure obtained from $\mathcal{A}$ by removing $q^{N}_{(i,j),k}$ equivalence classes in $N^{\mathcal{A}}_{(i,j),k}$.

Observe the following: If Spoiler selects a colored element from an equivalence class X in $\mathcal{A}$ and Duplicator responds by selecting another colored elements from an equivalence class Y in $\mathcal{B}$ such that $\mathrm{tp}(Y) \equiv^{C}_{n} \mathrm{tp}(X)$, there is no point for Spoiler to keep playing inside X because this will guarantee a win for Duplicator. Conversely, suppose Spoiler selects a colored element from an equivalence class X in $\mathcal{A}$, and there is no equivalence class in $\mathcal{B}$ whose type is colored n-equivalent to $\mathrm{tp}(X)$. Then Spoiler has a winning strategy for the game by playing inside X and Y.

Definition 3.4.3 *Consider game $G_n(\mathcal{A}, \mathcal{B})$ played on equivalence structures with one color. We say that a* colored disparity *occurs if there exists a type (i, j) and $n > k \geq 0$ such that the following holds:*

1. $k = q^{C}_{(i,j),n-k}$;
2. *In one of $\mathcal{A}^{C}((i, j), n - k)$ and $\mathcal{B}^{C}((i, j), n - k)$, there is an equivalence class whose type is colored $(n - k)$-equivalent to (i, j), and no such equivalence class exists in the other structure.*

We say that a non-colored disparity *occurs if there exists a type (i, j) and $k \in \{0, \cdots, n - 1\}$ such that the following holds:*

1. $k = q^N_{(i,j),n-k}$;
2. *In one of* $\mathcal{A}^N((i,j),n-k)$ *and* $\mathcal{B}^N((i,j),n-k)$, *there is an equivalence class whose type is non-colored* $(n-k)$*-equivalent to* (i,j), *and no such equivalence class exists in the other structure.*

Lemma 3.4.4 *Suppose* $\mathcal{A}$ *and* $\mathcal{B}$ *are two finite equivalence structures with one color. Duplicator wins the game* $G_n(\mathcal{A},\mathcal{B})$ *if and only if neither colored disparity nor non-colored disparity occurs in the game.*

Proof. Suppose a colored disparity occurs in game $G_n(\mathcal{A},\mathcal{B})$ as witnessed by (i,j) and k. Suppose in $\mathcal{A}^C((i,j),n-k)$ there is an equivalence class whose type is colored $(n-k)$-equivalent to (i,j), and no such equivalence class exists in the other structure. We describe a winning strategy for Spoiler. The case when a non-colored disparity occurs is treated in a similar manner. To win the game, Spoiler selects $k = q^C_{(i,j),n-k}$ pairwise non-equivalent elements $a_1,...,a_k$ in $\mathcal{A}$ from equivalence classes in $C^{\mathcal{A}}_{(i,j),n-k}$. Let $b_1,...,b_k$ be elements selected by Duplicator in response. For each $\ell \in \{1,\ldots,k\}$, let $[b_\ell]$ be the equivalence class of b_ℓ. Note that $\mathrm{tp}([b_\ell]) \equiv^C_{n-k} (i,j)$ as otherwise Spoiler would win. Now let X be an equivalence class in $\mathcal{A}$ such that $tp(X) \equiv^C_{n-k} (i,j)$ as stipulated in the lemma. Spoiler selects a colored element $x \in X$. Let y be the element selected by Duplicator in response to x and set $Y = [y]$. Note that $\mathrm{tp}(Y)$ cannot be colored $(n-k)$-equivalent to $\mathrm{tp}(X)$. By definition of $(n-k)$-equivalence, from now on, Spoiler uses a winning strategy inside X and Y and wins the game $G_n(\mathcal{A},\mathcal{B})$.

Conversely, suppose neither colored disparity nor non-colored disparity occurs in game $G_n(\mathcal{A},\mathcal{B})$, we then describe a strategy for Duplicator. Let us assume that the players have produced a k-round play $(a_1,b_1),(a_2,b_2),...,(a_k,b_k)$. Let (i_ℓ,j_ℓ) and (i'_ℓ,j'_ℓ) be respectively the types of a_ℓ and b_ℓ, with $1 \le \ell \le k$. Our inductive assumptions on this k-round play are the following:

(I1) For any $\ell \in \{1,\ldots,k\}$, a_ℓ is a colored element if and only if b_ℓ is a colored element.

(I2) For any $\ell, m \in \{1,\ldots,k\}$, $E(a_\ell,a_m)$ if and only if $E(b_\ell,b_m)$.

(I3) For any $\ell \in \{1,\ldots,k\}$, $(i_\ell,j_\ell) \equiv^C_{n-\ell} (i'_\ell,j'_\ell)$ and $(i_\ell,j_\ell) \equiv^N_{n-\ell} (i'_\ell,j'_\ell)$.

(I4) Let $\mathcal{A}'$ and $\mathcal{B}'$ be the equivalence structures obtained by removing equivalence classes $[a_1],...,[a_k]$ from $\mathcal{A}$ and $[b_1],...,[b_k]$ from $\mathcal{B}$, respectively. We assume in game $G_{n-k}(\mathcal{A}',\mathcal{B}')$ that neither colored disparity nor non-colored disparity occurs.

Assume that Spoiler selects an element $a_{k+1} \in A$. Duplicator responds to this move by choosing b_{k+1} as follows: If $a_{k+1} = a_l$ then $b_{k+1} = b_l$. Otherwise, if $E(a_{k+1},a_l)$ is true in $\mathcal{A}$, then Duplicator chooses a new b_{k+1} such that $E(b_{k+1},b_l)$ and a_{k+1} is a colored element if and only if b_{k+1} is a colored element. By (I3), Duplicator can always select such an element b_{k+1}.

Assume a_{k+1} is not equivalent to any of the elements $a_1, ..., a_k$. Let X be the equivalence class of a_{k+1} in $\mathcal{A}$. If a_{k+1} is a colored element, then Duplicator chooses a colored element b_{k+1} from an equivalence class Y of $\mathcal{B}$ such that $\text{tp}(X) \equiv^C_{n-k} \text{tp}(Y)$. If a_{k+1} is a non-colored element, then Duplicator chooses a non-colored b_{k+1} from an equivalence class Y of $\mathcal{B}$ such that $\text{tp}(X) \equiv^N_{n-k} \text{tp}(Y)$. Note that such an equivalence class Y must exist in $\mathcal{B}$, as otherwise either colored or non-colored disparity would occur in $G_{n-k}(\mathcal{A}', \mathcal{B}')$ as witnessed by $\text{tp}(X)$ and 0. The case when Spoiler selects an element from B is treated in a similar manner.

On the play $(a_1, b_1), ..., (a_k, b_k), (a_{k+1}, b_{k+1})$, inductive assumption (I1) and (I2) can be easily checked to hold. To prove that inductive assumption (I3) holds, let (i_{k+1}, j_{k+1}) and (i'_{k+1}, j'_{k+1}) be the type of $[a_{k+1}]$ and $[b_{k+1}]$, respectively. The strategy ensures one of $(i_{k+1}, j_{k+1}) \equiv^C_{n-k}$ (i'_{k+1}, j'_{k+1}) and $(i_{k+1}, j_{k+1}) \equiv^N_{n-k} (i'_{k+1}, j'_{k+1})$ is true, and by Lemma 3.4.2, $(i_{k+1}, j_{k+1}) \equiv^C_{n-k-1}$ (i'_{k+1}, j'_{k+1}) and $(i_{k+1}, j_{k+1}) \equiv^N_{n-k-1} (i'_{k+1}, j'_{k+1})$.

It remains for us to prove that inductive assumption (I4) is preserved. Consider the structure $\mathcal{A}''$ and $\mathcal{B}''$ obtained by removing $[a_1], ..., [a_{k+1}]$ from $\mathcal{A}$ and $[b_1], ..., [b_{k+1}]$ from $\mathcal{B}$, respectively. Suppose a colored disparity occurs in $G_{n-k-1}(\mathcal{A}'', \mathcal{B}'')$ as witnessed by some type (i, j) and $t \in \{0, \ldots, n-k-2\}$. There are two cases. If $(i, j) \equiv^C_{n-k-t-1} (i_{k+1}, j_{k+1})$, then by Lemma 3.4.2, $(i, j) \equiv^C_{n-k-t-1} (i'_{k+1}, j'_{k+1})$, and a colored disparity occurs in $G_{n-k}(\mathcal{A}', \mathcal{B}')$ as witnessed by (i, j) and $t+1$; If $(i, j) \not\equiv^C_{n-k-t-1} (i_{k+1}, j_{k+1})$, then by Lemma 3.4.2, $(i, j) \not\equiv^C_{n-k-t-1}$ (i'_{k+1}, j'_{k+1}), and a colored disparity occurs in $G_{n-k}(\mathcal{A}', \mathcal{B}')$ as witnessed by (i, j) and t, contradicting our assumption. The case when a non-colored disparity occurs in $G_{n-k-1}(\mathcal{A}'', \mathcal{B}'')$ is treated in a similar way.

Hence the strategy is a winning strategy for Duplicator by inductive assumptions (1) and (2). The lemma is proved. ∎

Theorem 3.4.5 *Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins game $G_n(\mathcal{A}, \mathcal{B})$ on finite equivalence structures with one color $\mathcal{A}$ and $\mathcal{B}$. The constant that bounds the running time is n^3.*

Proof. This result follows from the lemmas above. We present colored equivalence structures $\mathcal{A}$ in three lists. The first one lists equivalence classes of $\mathcal{A}$ in increasing order of their types; the second and the third one list the sequence $\{q^{\mathcal{A},C}_{(i,j),k}\}_{0 \le i,j,k \le n}$ and $\{q^{\mathcal{A},N}_{(i,j),k}\}_{0 \le i,j,k \le n}$ respectively. The algorithm checks whether a colored or a non-colored disparity occurs by reading the second and third list. If the algorithm detects a disparity then Spoiler wins, otherwise, Duplicator wins. The running time for the process is bounded by n^3. ∎

Fix $s > 1$, let $\mathcal{A}$ be an equivalence structure with s colors. For each element x of $\mathcal{A}$, define the *characteristic* of x, $\text{ch}(x)$, as a binary sequence $t_1 t_2 ... t_s \in \{0, 1\}^s$ such that for each $i \in \{1, \ldots, s\}$, $t_i = 1$ if $x \in P_i$ and $t_i = 0$ otherwise. There are 2^s pairwise distinct characteristics, and we order them in lexicographic order: $c_1, ..., c_{2^s}$. Construct the structure $\mathcal{A}' = (A; E, Q_1, ..., Q_{2^s})$ such that for all $1 \le i \le 2^s$, $Q_i = \{x \in A \mid ch(x) = c_i\}$.

The following is an easy lemma:

Lemma 3.4.6 *Let $\mathcal{A} = (A; E, P_1, ..., P_s)$ be an equivalence structure with s unary predicates.*

1. *For any two distinct characteristics c_i and c_j, we have $Q_i \cap Q_j = \emptyset$.*
2. *$\mathcal{A}$ and $\mathcal{B}$ are isomorphic if and only if $\mathcal{A}'$ and $\mathcal{B}'$ are isomorphic.*

For an equivalence class X, we define the *type* of X, tp(X), as a tuple $(i_1, i_2, ..., i_{2^s}) \in \mathbb{N}^{2^s}$ such that in X, the number of elements with characteristic c_j is i_j for all $1 \leq j \leq 2^s$; thus $\sum_{j=1}^{2^s} i_j = |X|$.

Definition 3.4.7 *Let $\kappa = (i_1, ..., i_{2^s})$ and $\lambda = (i'_1, ..., i'_{2^s})$ be two types of equivalence classes. For $1 \leq j \leq 2^s$, we say that κ is (j, n)-*equivalent *to λ, denoted by $\kappa \equiv_n^j \lambda$, if the following holds.*

1. *If $i_j < n$ then $i'_j = i_j$, otherwise $i'_j \geq n$; and*
2. *For all $1 \leq \ell \leq 2^s$ where $\ell \neq j$, if $i_\ell < n - 1$ then $i'_\ell = i_\ell$, otherwise $i'_\ell \geq n - 1$.*

Let X and Y be equivalence classes of types κ and λ respectively. If $\kappa \equiv_n^j \lambda$, then Duplicator wins the n-round EF game played on structures $(X; E \restriction X, P_1 \restriction X, ..., P_s \restriction X)$ and $(Y; E \restriction Y, P_1 \restriction Y, ..., P_s \restriction Y)$, given that Spoiler selects an element $x \in X$ with characteristic c_j.

Similar to the case of equivalence structures with one color, we introduce the following notions:

– For type λ, $1 \leq j \leq 2^s$ and $k \geq 1$, we set $C_{\lambda,k}^{\mathcal{A},j}$ as the set

$$\{X \mid X \text{ is an equivalence class of } \mathcal{A} \text{ and } \text{tp}(X) \equiv_k^j \lambda\}.$$

– For type λ, $1 \leq j \leq 2^s$ and $k \geq 1$, set

$$q_{\lambda,k}^{\mathcal{A},j} = |C_{\lambda,k}^{\mathcal{A},j}|$$

– For $\mathcal{A}$ and $\mathcal{B}$, set

$$q_{\lambda,k}^{j} = \min\{q_{\lambda,k}^{\mathcal{A},j}, q_{\lambda,k}^{\mathcal{B},j}\}$$

– Set $\mathcal{A}^j(\lambda, k)$ as the structure obtained from $\mathcal{A}$ by removing $q_{\lambda,k}^{j}$ equivalence classes in $C_{\lambda,k}^{\mathcal{A},j}$.

Definition 3.4.8 *Consider game $G_n(\mathcal{A}, \mathcal{B})$ played on equivalence structures with s colors. For $1 \leq j \leq 2^s$, we say that a* disparity occurs with respect to *c_j if there exists a type $\lambda = (i_1, ..., i_{2^s})$ and $n > k \geq 0$ such that the following holds:*

1. $k = q^j_{\lambda,n-k}$;

2. *In one of* $\mathcal{A}^j(\lambda, n-k)$, *there is an equivalence class whose type is* $(j, n-k)$*-equivalent to* λ, *and no such equivalence class exists in the other structure.*

Essentially the same proof of Lemma 3.4.4 can be used to prove the following lemma.

Lemma 3.4.9 *Suppose* $\mathcal{A}$ *and* $\mathcal{B}$ *are two equivalence structures with s colors. Duplicator wins the game* $G_n(\mathcal{A}, \mathcal{B})$ *if and only if disparity does not occur with respect to* c_j *for any* $1 \leq j \leq 2^s$.

By the lemma above, we can extend Theorem 3.4.5 to the following results.

Theorem 3.4.10 *Fix* $n \in \omega$. *There exists an algorithm that runs in constant time and decides whether Duplicator wins the game* $G_n(\mathcal{A}, \mathcal{B})$ *on finite equivalence structures with s colors. The constant that bounds the running time is* n^{2^s+1}.

3.5 Embedded equivalence structures

In this section we study *embedded equivalence structure of height h*; these are structures of the form $\mathcal{A} = (A; E_1, E_2, ..., E_h)$ such that each E_i where $1 \leq i \leq h$ is an equivalence relation and $E_i \subseteq E_j$ for $i < j$. Such hierarchical structures may appear as models of university or large company databases. For example, in a university database there could be the SameFaculty and SameDepartment relations. The first relation stores all tuples (x, y) such that x and y belong to the same faculty; similarly, the second relation stores all tuples (u, v) such that u and v are in the same department. These relations are equivalence relations. Moreover, a natural connection between these two relations is as sets the relation SameDepartment is a subset of the relation SameFaculty. In this section, we give a full solution for EF games played on embedded equivalence structures of height h. We start with the case where $h = 2$. The case for $h > 2$ will be explained later.

Let $\mathcal{A} = (A; E_1, E_2)$ be a finite embedded equivalence structure of height 2. We say that an E_2-equivalence class X has *type* $\mathsf{tp}(X) = (q_1, \ldots, q_t)$ if the largest E_1-equivalence class contained in X has size t and for all $1 \leq i \leq t$, q_i is the number of E_1-equivalence classes of size i contained in X. Thus, $\sum_{i=1}^{t}(q_i \times i) = |X|$. For two types $\sigma = (q_1, \ldots, q_{t_1})$ and $\tau = (q'_1, \ldots, q'_{t_2})$, we say $\sigma = \tau$ if $t_1 = t_2$ and $q_i = q'_i$ for all $i \in \{1, \ldots, t_1\}$.

For $X \subseteq A$, we use $(X; E_1 \restriction X)$ to denote the equivalence structure obtained by restricting E_1 on X. Given two E_2-equivalence classes X and Y of types σ and τ respectively, we say that σ is *n-equivalent* to τ, denoted by $\sigma \equiv_n \tau$, if Duplicator wins the n-round game played on structures $(X; E_1 \restriction X)$ and $(Y; E_1 \restriction Y)$. Note that if $\sigma \equiv_n \tau$, then $\sigma \equiv_i \tau$ for all $i \leq n$.

We need the following notations:

– For type σ and $i \geq 1$, set

$$C^{\mathcal{A}}_{\sigma,i} = \{X \mid X \text{ is an } E_2\text{-equivalence class of } \mathcal{A} \ \wedge \ \mathrm{tp}(X) \equiv_i \sigma\}.$$

– Set $q^{\mathcal{A}}_{\sigma,i} = |C^{\mathcal{A}}_{\sigma,i}|$.

– For finite embedded equivalence structure $\mathcal{A}$ and $\mathcal{B}$, set $q^{\sigma,i} = \min\{q^{\mathcal{A}}_{\sigma,i}, q^{\mathcal{B}}_{\sigma,i}\}$.

– Set $\mathcal{A}(\sigma, i)$ be the embedded equivalence structure of height 2 obtained from $\mathcal{A}$ by removing $q^{\sigma,i}$ equivalence classes whose types are i-equivalent to σ.

Observe that in round k of game $G_n(\mathcal{A}, \mathcal{B})$, if Spoiler selects an element from an E_2-equivalence class X in $\mathcal{A}$, and Duplicator responds by selecting another element from an E_2-equivalence class Y in $\mathcal{B}$ such that $\mathrm{tp}(Y) \equiv_{n-k} \mathrm{tp}(X)$, there is no reason for Spoiler to keep playing inside X because this will guarantee a win for Duplicator. Intuitively, $\mathcal{A}(\sigma, n-k)$ contains all the E_2-equivalence classes for Spoiler to choose elements from after $q^{\sigma,n-k}$ many E_2-equivalence classes whose types are $(n-k)$-equivalent to σ have been chosen.

Definition 3.5.1 *Consider a game $G_n(\mathcal{A}, \mathcal{B})$ played on finite embedded equivalence structures of height 2. We say that a* disparity *occurs if there exists a type σ and $n > k \geq 0$ such that the following holds.*

1. $k = q^{\sigma,n-k}$.
2. *In one of $\mathcal{A}(\sigma, n-k)$ and $\mathcal{B}(\sigma, n-k)$, there is an E_2-equivalence class whose type is $(n-k)$-equivalent to σ, and no such E_2-equivalence class exists in the other structure.*

Lemma 3.5.2 *Suppose $\mathcal{A}$ and $\mathcal{B}$ are two finite embedded equivalence structures of height 2. Duplicator wins the game $G_n(\mathcal{A}, \mathcal{B})$ if and only if no disparity occurs.*

Proof. Suppose disparity occurs in game $G_n(\mathcal{A}, \mathcal{B})$ as witnessed by σ and k, in $\mathcal{A}(\sigma, n-k)$ there is an E_2-equivalence class whose type is $(n-k)$-equivalent to σ, and no such E_2-equivalence class exists in $\mathcal{B}(\sigma, n-k)$. We describe a winning strategy for Spoiler as follows: Spoiler selects $k = q^{\sigma,n-k}$ pairwise non-E_2-equivalent elements $a_1, \ldots, a_k$ in $\mathcal{A}$ from E_2-equivalence classes whose types are $(n-k)$-equivalent to σ. Let $b_1, \ldots, b_k$ be elements selected by Duplicator in response. For each $i \in \{1, \ldots, k\}$, let $[b_i]_{E_2}$ be E_2-equivalence class that b_i is in. Note that $\mathrm{tp}([b_i]) \equiv_{n-k} \sigma$ as otherwise Spoiler would win. Now let X be an equivalence class in $\mathcal{A}$ such that $\mathrm{tp}(X) \equiv_{n-k} \sigma$ as stipulated in the lemma. Spoiler selects an element $x \in X$. Let y be the element selected by Duplicator in response to x and set $Y = [y]_{E_2}$. Note that $\mathrm{tp}(Y)$ cannot be $(n-k)$-equivalent to $\mathrm{tp}(X)$. By definition of $(n-k)$-equivalence, henceforth Spoiler uses a winning strategy inside X and Y and wins game $G_n(\mathcal{A}, \mathcal{B})$.

Conversely, suppose no disparity occurs in the game $G_n(\mathcal{A}, \mathcal{B})$, we then describe a strategy for Duplicator. Let us assume that the players have produced a k-round play $(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)$. Let σ_i and τ_i be the types of a_i and b_i, respectively with $1 \leq i \leq k$. Our inductive assumptions on this k-round play are the following:

1. The map $a_i \to b_i$ is partial isomorphism.
2. For all $1 \leq i \leq k$, $\sigma_i \equiv_{n-i} \tau_i$.
3. Let $\mathcal{A}'$ and $\mathcal{B}'$ be the equivalence structures obtained by removing the E_2-equivalence classes $[a_1]_{E_2}, \ldots, [a_k]_{E_2}$ from $\mathcal{A}$ and the equivalence classes $[b_1]_{E_2}, \ldots, [b_k]_{E_2}$ from $\mathcal{B}$, respectively. We assume in game $G_{n-k}(\mathcal{A}', \mathcal{B}')$ that no disparity occurs.

Assume that Spoiler selects an element $a_{k+1} \in A$. Duplicator responds to this move by choosing b_{k+1} as follows. If $a_{k+1} = a_i$ then $b_{k+1} = b_i$. Otherwise, if $E_1(a_i, a_{k+1})$ is true in $\mathcal{A}$, then Duplicator chooses a new b_{k+1} such that $E_1(b_i, b_{k+1})$. If $E_2(a_i, a_{k+1})$ is true in $\mathcal{A}$ and there is no j such that $E_1(a_j, a_{k+1})$, then Duplicator chooses a new b_{k+1} such that $E_2(b_i, b_{k+1})$ and there is no j such that $E_1(b_j, b_{k+1})$. By (2) of the inductive assumption Duplicator can always select such an element b_{k+1} by following its winning strategies.

Assume a_{k+1} is not equivalent to any of the elements $a_1, ..., a_k$. Let X be the E_2-equivalence class in $\mathcal{A}$ that contains a_{k+1}. Duplicator selects b_{k+1} from an E_2-equivalence class Y in $\mathcal{B}$ such that $\text{tp}(X) \equiv_{n-k} \text{tp}(Y)$. Duplicator is able to select such an element as otherwise disparity would occur as witnessed by the type of X and 0.

The case when Spoiler selects an element from B is treated similarly.

Inductive assumption (1) and (2) can be easily checked to hold on the play $(a_1, b_1), \ldots, (a_k, b_k), (a_{k+1}, b_{k+1})$. To show that assumption (3) is preserved, consider the structures $\mathcal{A}''$ and $\mathcal{B}''$ obtained by removing $[a_1]_{E_2}, \ldots, [a_k]_{E_2}, [a_{k+1}]_{E_2}$ and $[b_1]_{E_2}, \ldots, [b_k]_{E_2}, [b_{k+1}]_{E_2}$ from $\mathcal{A}$ and $\mathcal{B}$, respectively. Suppose disparity occurs in $G_{n-k-1}(\mathcal{A}'', \mathcal{B}'')$ as witnessed by some type τ and $t < n - k - 1$. There are two cases. If $\text{tp}([a_{k+1}]) \equiv_{n-k-t-1} \tau$, then $\text{tp}([b_{k+1}]) \equiv_{n-k-t-1} \tau$, and disparity must occur in $G_{n-k}(\mathcal{A}', \mathcal{B}')$ as witnessed by τ and $t+1$. If $\text{tp}([a_{k+1}]) \not\equiv_{n-k-t-1} \tau$, then $\text{tp}([b_{k+1}]) \not\equiv_{n-k-t-1} \tau$, and disparity must occur in $G_{n-k}(\mathcal{A}', \mathcal{B}')$ as witnessed by τ and t, contradicting our assumption. Hence the strategy is a winning strategy for Duplicator by inductive assumption (1). ∎

Theorem 3.5.3 *Fix $n \in N$. There exists an algorithm that runs in constant time and decides whether Duplicator wins game $G_n(\mathcal{A}, \mathcal{B})$ on finite embedded equivalence structures of height 2. The constant that bounds the running time is $(n + 1)^n$.*

Proof. This result follows from Lemma 3.5.2. We represent structure $\mathcal{A} = (A; E_1, E_2)$ by a tree and a list. The tree has height 3. The leaves of the tree are all elements in A. Two

leaves x, y have the same parent if $E_1(x, y)$, and x, y have the same ancestor at level 1 if $E_2(x, y)$. Intuitively, we can view the root of tree as A, the internal nodes at level 1 represent all E_2-equivalence classes on A, and the children of each E_2-equivalence class X at level 2 are all E_1-equivalence classes contained in X. We further require that representations of E_2 and E_1-equivalence classes are put in left-to-right order according to their cardinalities.

The list is $q^{\mathcal{A}}_{\sigma_1,1}, ..., q^{\mathcal{A}}_{\sigma_t,1}, q^{\mathcal{A}}_{\sigma_1,2}, ..., q^{\mathcal{A}}_{\sigma_1,2}, ..., q^{\mathcal{A}}_{\sigma_1,n}, ..., q^{\mathcal{A}}_{\sigma_t,n}$ where each σ_i is a type of E_2-equivalence class, and $q^{\mathcal{A}}_{\sigma_i,j}$ is as defined above. Each $q^{\mathcal{A}}_{\sigma_i,j}$ has a value between 0 and n and if it is greater than n, we set it to n.

The algorithm checks whether disparity occurs in $G_n(\mathcal{A}, \mathcal{B})$ by examining the list. There can be at most $(n + 1)^n$ pairwise non-n-equivalent types. Therefore, checking disparity requires a time bounded by $(n + 1)^{n+1}$. ∎

For the case when $\mathcal{A}$ and $\mathcal{B}$ are two embedded equivalence structures of height h, where $h > 2$, we give a similar definition of the type of an E_h-equivalence class. We can then describe the winning conditions for Spoiler and Duplicator in a similar way.

Let $\mathcal{A}$ be an embedded equivalence structure of height h where $h > 2$. For an E_h-equivalence class X, we recursively define $\mathrm{tp}(X)$, the *type* of X. Set $\mathrm{tp}(X)$ be $(q_{\sigma_1}, \ldots, q_{\sigma_t})$ that satisfies the following properties.

1. Each σ_i is the type of an E_{h-1}-equivalence class.

2. σ_t is the maximum type in lexicographic order among all types of E_{h-1}-equivalence classes contained in X.

3. The list $\sigma_1, ..., \sigma_t$ contains all possible types of E_{h-1}-equivalence classes less or equal to σ_t ordered lexicographically.

4. For all $1 \leq i \leq t$, q_{σ_i} is the number of all E_{h-1}-equivalence classes contained in X whose type are σ_i.

Let $\kappa = (q_{\sigma_1}, ..., q_{\sigma_s})$ and $\lambda = (q'_{\sigma_1}, ..., q'_{\sigma_t})$ be types of two E_h-equivalence classes X and Y, respectively. We say $\kappa = \lambda$ if $s = t$ and $q_{\sigma_i} = q'_{\sigma_i}$ for all $i \in \{1, \ldots, s\}$. We say $\kappa \equiv_n \lambda$ if the structures $(X; E_1 \restriction X, \ldots, E_{h-1} \restriction X)$ and $(Y; E_1 \restriction Y, \ldots, E_{h-1} \restriction Y)$ are n-equivalent.

The following proposition shows that $\mathrm{tp}(X)$ are isomorphism invariants of the E_h-equivalence classes.

Proposition 3.5.4 *Let X and Y be two E_h-equivalence classes in a finite embedded equivalence structure $\mathcal{A} = (A; E_1, ..., E_h)$. Then $tp(X) = \mathrm{tp}(Y)$ if and only if the structures $(X; E_1 \restriction X, \ldots, E_{h-1} \restriction X)$ and $(Y; E_1 \restriction Y, \ldots, E_{h-1} \restriction Y)$ are isomorphic. In particular, the isomorphism problem for embedded equivalence structure of height h is linear on the size of the structure.*

Proof. The first part of the proposition easily follows from the definition of the types. To prove the second part of the proposition, suppose $\mathcal{A}$ and $\mathcal{B}$ are two embedded equivalence structures of height h. We represent them by listing E_h-equivalence classes in a manner that their types are lexicographically ordered. Suppose $\mathcal{A}$ and $\mathcal{B}$ are represented by listing E_h-equivalence classes $X_1, X_2, \ldots, X_{k_1}$ and $Y_1, Y_2, \ldots, Y_{k_2}$, respectively. Then $\mathcal{A}$ and $\mathcal{B}$ are isomorphic if and only if $k_1 = k_2$ and for all $1 \leq i \leq k_1$, $(X_i; E_1 \restriction X_i) \cong (Y_i; E_1 \restriction Y_i)$, which is same as $\mathrm{tp}(X_i) = \mathrm{tp}(Y_i)$. ∎

Similarly to the case of embedded equivalence structures of height 2, we re-introduce the notions $C^{\mathcal{A}}_{\sigma,i}$, $q^{\mathcal{A}}_{\sigma,i}$, $q^{\sigma,i}$, $\mathcal{A}(\sigma, i)$ and disparity in game $G_n(\mathcal{A}, \mathcal{B})$. The only difference would be that in the new definition, we refer to the E_h-equivalence classes wherever we refer to E_2-equivalence classes in the original definition. The following lemma can thus be proved in a similar manner as Lemma 3.5.2.

Lemma 3.5.5 *Suppose $\mathcal{A}$ and $\mathcal{B}$ are two finite embedded equivalence structures of height h where $h \geq 2$. Duplicator wins game $G_n(\mathcal{A}, \mathcal{B})$ if and only if no disparity occurs.*

A simple calculation reveals that the number of pairwise non-n-equivalent types of E_h-equivalence classes is at most $(n+1)^{\cdots^{(n+1)^n}}$ where the tower of $(n+1)$ has height h. Therefore, by Lemma 3.5.5, we can extend Theorem 3.5.3 to the following result.

Theorem 3.5.6 *Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins game $G_n(\mathcal{A}, \mathcal{B})$ on finite embedded equivalence structures of height h $\mathcal{A} = (A; E_1, ..., E_h)$ and $\mathcal{B} = (B; E_1, ..., E_h)$. The constant that bounds the running time is $< (n+1)^{\cdots^{(n+1)^{(n+1)}}}$ where the tower of $(n+1)$ has height h.*

3.6 Trees with level predicates

In this section we study EF games played on *trees with level predicates*; these are structures of the type $\mathcal{T} = (T; \leq, L_0, \ldots, L_h)$, where $(T; \leq)$ is a tree of height h, and for $0 \leq i \leq h$, L_i is a unary predicate such that an element $x \in T$ belongs to L_i if and only if x has level i. We fix number $h \geq 2$ and restrict ourselves to the class $\mathfrak{K}_h$ of finite trees with level predicates of height at most h. Deciding EF games on trees from $\mathfrak{K}_h$ can be done directly by using the techniques from the previous section. Instead, we reduce the problem of deciding EF games on trees with level predicates in $\mathfrak{K}_h$ to one for embedded equivalence structures of height $h + 1$.

We transform trees from the class $\mathfrak{K}_h$ into the class of embedded equivalence structures of height h in the following manner. Let $\mathcal{T}$ be a tree in $\mathfrak{K}_h$. We now define an embedded equivalence structure $\mathcal{A}(\mathcal{T})$ as follows. The domain A of $\mathcal{A}(\mathcal{T})$ is now $T \cup \{a_x \mid x$ is a leaf of

$\mathcal{T}\}$. We define the equivalence relation E_i, $1 \leq i \leq h$, on the domain as follows. The relation E_1 is the minimal equivalence relation that contains $\{(x, a_x) \mid x \text{ is a leaf of } \mathcal{T}\}$. Let $x_1, \ldots,$ x_s be all elements of $\mathcal{T}$ at level $h - i + 1$ where $1 \leq i < h$. Let $\mathcal{T}_1, \ldots, \mathcal{T}_s$ be the subtrees of $\mathcal{T}$ whose roots are $x_1, \ldots, x_s$, respectively. Set E_i be the minimal equivalence relation that contains $E_{i-1} \cup T_1^2 \cup \ldots \cup T_s^2$. It is clear that $E_i \subseteq E_{i+1}$ for all $1 \leq i \leq h$. Thus we have the embedded equivalence structure $\mathcal{A}(T) = (A; E_1, ..., E_h)$.

Lemma 3.6.1 *For trees $\mathcal{T}_1$ and $\mathcal{T}_2$, $\mathcal{T}_1 \cong \mathcal{T}_2$ if and only if $\mathcal{A}(\mathcal{T}_1) \cong \mathcal{A}(\mathcal{T}_2)$. In particular, Duplicator wins game $G_n(\mathcal{T}_1, \mathcal{T}_2)$ if and only if Duplicator wins $G_n(\mathcal{A}(\mathcal{T}_1), \mathcal{A}(\mathcal{T}_2))$.*

Proof. Suppose $\mathcal{T}$ is a tree in the class $\mathfrak{K}_h$. Take an element $x \in T$. By construction of $\mathcal{A}(\mathcal{T})$, the following statements are true.

- x is a leaf in $\mathcal{T}$ if and only if $|\{y \mid E_1(x, y)\}| = 2$ in $\mathcal{A}(\mathcal{T})$.
- x is the root of $\mathcal{T}$ if and only if $|\{y \mid E_h(x, y)\}| = 1$ in $\mathcal{A}(\mathcal{T})$.

We define the *level* of x in $\mathcal{A}(\mathcal{T})$ as follows. If x is the root of $\mathcal{T}$, the level of x is 0. Otherwise, if x is an internal node, the level of x in $\mathcal{A}(\mathcal{T})$ is the largest ℓ such that $|\{y \mid E_{h-\ell+1}(x, y)\}| > 1$. If x is a leaf, we define the level of x in $\mathcal{A}(\mathcal{T})$ to be the largest $\ell + 1$ such that there is an internal node y such that $E_{h-l+1}(x, y)$.

By definition, for all $x \in T$, the level of x in $\mathcal{T}$ coincides with the level of x in $\mathcal{A}(\mathcal{T})$. For $x, y \in T$, $x \leq y$ in $\mathcal{T}$ if and only in $\mathcal{A}(\mathcal{T})$ x has level s and y has level t such that $s \geq t$ and $E_{h-t+1}(x, y)$. Therefore given two trees from $\mathfrak{K}_h$, $\mathcal{T}_1$ and $\mathcal{T}_2$, and a mapping $f : T_1 \to T_2$, f is an isomorphism between $\mathcal{T}_1$ and $\mathcal{T}_2$ if and only if f is an isomorphism between $\mathcal{A}(\mathcal{T}_1)$ and $\mathcal{A}(\mathcal{T}_2)$.

To prove the second part of the lemma, assume that there is a winning strategy for Spoiler on game $G_n(\mathcal{T}_1, \mathcal{T}_2)$. It is easy to see that this strategy is also a winning strategy for Spoiler on game $G_n(\mathcal{A}(\mathcal{T}_1), \mathcal{A}(\mathcal{T}_2))$, as otherwise Duplicator would win the game $G_n(\mathcal{T}_1, \mathcal{T}_2)$.

Conversely, assume that there is a winning strategy for Duplicator on the n-round game $G_n(\mathcal{T}_1, \mathcal{T}_2)$. We describe a strategy for Duplicator on the game $G_n(\mathcal{A}(\mathcal{T}_1), \mathcal{A}(\mathcal{T}_2))$ where $\mathcal{A}(\mathcal{T}_1) = (A_1; E_1, ..., E_h)$ and $\mathcal{A}(\mathcal{T}_2) = (A_2; E_1, ..., E_h)$. Let us assume that the players have produced a k-round play $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$. Assume on this k-round play that the map $x_i \to y_i$ is a partial isomorphism between $\mathcal{A}(\mathcal{T}_1)$ and $\mathcal{A}(\mathcal{T}_2)$.

Assume that Spoiler selects an element $x_{k+1} \in A_1$. Duplicator responds to this move by choosing x_{k+1} as follows. If $x_{k+1} = x_i$ then $y_{k+1} = y_i$. Otherwise, if $x_{k+1} \in T_1$, then Duplicator selects an element $y_{k+1} \in T_2$ according to its winning strategy on $G_n(\mathcal{T}_1, \mathcal{T}_2)$. If $x_{k+1} = a_x$ for some leaf $x \in \mathcal{T}_1$, then Duplicator responds by selecting $y_{k+1} = a_y$ where y is the leaf in T_2 that corresponds to x in Duplicator's winning strategy in $G_n(\mathcal{T}_1, \mathcal{T}_2)$. It is clear that $x_i \to y_i$ where $1 \leq i \leq k + 1$ is also a partial isomorphism between $\mathcal{A}(\mathcal{T}_1)$ and $\mathcal{A}(\mathcal{T}_2)$. Therefore the strategy described is a winning strategy for Duplicator on game $G_n(\mathcal{A}(\mathcal{T}_1), \mathcal{A}(\mathcal{T}_2))$. ∎

Theorem 3.6.2 *Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins game $G_n(\mathcal{T}_1, \mathcal{T}_2)$ on finite trees with level predicates $\mathcal{T}_1$ and $\mathcal{T}_2$ from the class $\mathfrak{K}_h$. The constant that bounds the running time is $< (n+1)^{\cdots^{(n+1)^{(n+1)}}}$ where the tower has height h.*

Proof. To prove the theorem, the trees with level predicates $\mathcal{T}_1$ and $\mathcal{T}_2$ are represented by the two embedded equivalence structures of height h $\mathcal{A}(\mathcal{T}_1)$ and $\mathcal{A}(\mathcal{T}_2)$, respectively. By Lemma 3.6.1, Duplicator wins game $G_n(\mathcal{A}(\mathcal{T}_1), \mathcal{A}(\mathcal{T}_2))$ if and only if Duplicator wins game $G_n(\mathcal{T}_1, \mathcal{T}_2)$. By Theorem 3.5.6, we have a constant time algorithm that decides the winner of the EF game $G_n(\mathcal{A}(\mathcal{T}_1), \mathcal{A}(\mathcal{T}_2))$ in which the constant is bounded by $(n+1)^{\cdots^{(n+1)^{(n+1)}}}$ where the tower has height h. ∎

3.7 Boolean algebras with distinguished ideals

In this section we study EF games played on *Boolean algebras with distinguished ideals*; these are structures of the form $\mathcal{A} = (A; \leq, 0, 1, I_1, \ldots, I_s)$, where $(A; \leq, 0, 1)$ forms a Boolean algebra and each I_j is an ideal of the algebra $(A; \leq, 0, 1)$. The set of **atoms** of $\mathcal{A}$, denoted $\mathrm{At}(\mathcal{A})$, is the set $\{a \mid \forall y : 0 \leq y \leq a \rightarrow y = 0 \vee y = a\}$. Since we restrict ourselves to finite structures, the Boolean algebra $(A; \leq, 0, 1)$ can be identified with the structure $(2^{X_A}; \subseteq, \emptyset, X_A)$, where $X_A = \mathrm{At}(\mathcal{A})$ and 2^{X_A} is the collection of all subsets of X_A. Moreover, for each ideal I_j there exists a set $A_j \subset \mathrm{At}(\mathcal{A})$ such that $I_j = 2^{A_j}$. Hence the original structure $\mathcal{A}$ can be identified with the following structure:

$$(2^{X_A}; \subseteq, \emptyset, X_A, 2^{A_1}, \ldots, 2^{A_s}).$$

For each element $x \in \mathrm{At}(\mathcal{A})$, define the *characteristic* of x, $\mathrm{ch}(x)$, as a binary word $t_1 t_2 ... t_s \in \{0,1\}^s$ such that for each $i \in \{1, \ldots, s\}$, $t_i \in \{0,1\}$ and $t_i = 1$ if and only if $x \in A_i$. For each characteristic $\epsilon \in \{0,1\}^s$ consider the set $A_\epsilon = \{x \in \mathrm{At}(\mathcal{A}) \mid \mathrm{ch}(x) = \epsilon)\}$. This defines the ideal I_ϵ in the Boolean algebra $(2^{X_A}; \subseteq, \emptyset, X_A)$. Moreover, we can also identify this ideal with the Boolean algebra $(2^{A_\epsilon}; \subseteq, \emptyset, A_\epsilon)$. There are 2^s pairwise distinct characteristics. Let $\epsilon_1, \ldots, \epsilon_{2^s}$ be the list of all characters. We denote by $\mathcal{A}'$ the following structure:

$$(2^X; \subseteq, \emptyset, X, 2^{A_{\epsilon_1}}, \ldots, 2^{A_{\epsilon_{2^s}}}).$$

The following is an easy lemma:

Lemma 3.7.1 *Let $\mathcal{A} = (2^{X_A}; \subseteq, \emptyset, X_A, 2^{A_1}, \ldots, 2^{A_s})$ be a Boolean algebra with distinguished ideals.*

1. *For any two distinct characteristics ϵ and δ we have $I_\epsilon \cap I_\delta = \{\emptyset\}$.*
2. *For any element $a \in 2^{X_A}$ there are elements $a_{\epsilon_i} \in I_{\epsilon_i}$ for $i \in \{1, \ldots, 2^s\}$ such that $a = \cup_{1 \leq i \leq 2^s} a_{\epsilon_i}$.*

3. *The Boolean algebra* $(2^{X_A}; \subseteq, \emptyset, X_A)$ *is isomorphic to the Cartesian product of the Boolean algebras* I_{ϵ_i}, $1 \leq i \leq 2^s$.

4. $\mathcal{A}$ *and* $\mathcal{B}$ *are isomorphic if and only if* $\mathcal{A}'$ *and* $\mathcal{B}'$ *are isomorphic.*

The next lemma connects the structure $\mathcal{A}'$ and $\mathcal{A}$ in terms of characterizing the winner of the game $G_n(\mathcal{A}, \mathcal{B})$.

Lemma 3.7.2 *Duplicator wins the game* $G_{n+1}(\mathcal{A}, \mathcal{B})$ *if and only if each of the following two conditions are true:*

1. *For each characteristic* ϵ, $|A_\epsilon| \geq 2^n$ *if and only if* $|B_\epsilon| \geq 2^n$.

2. *For each characteristic* ϵ, *if* $|A_\epsilon| < 2^n$ *then* $|A_\epsilon| = |B_\epsilon|$.

Proof. Assume that for some ϵ, we have $|A_\epsilon| \neq |B_\epsilon|$ and $|B_\epsilon| < 2^n$. Let us assume that $|A_\epsilon| \geq 2^n$. The case when $|A_\epsilon| < 2^n$ is treated in a similar manner. We describe a winning strategy for Spoiler. Spoiler starts by taking elements $a_1, a_2, \ldots$ in A_ϵ. For each $i \leq n$ the element a_i is such that $|\mathrm{At}(a_i)| \geq 2^{n-i}$ where $\mathrm{At}(a)$ denotes the set of atoms below a. The elements $a_1, a_2, \ldots$ are such that for each i, either $a_i \subset a_{i-1}$ or $a_i \cap a_{i-1} = \emptyset$. Consider the k round play $(a_1, b_1), \ldots, (a_k, b_k)$ where $k < n$. Let $e < k$ be the last round for which $a_k \subset a_e$. If no such e exists, let $a_e = 2^{A_\epsilon}$ and $b_e = 2^{B_\epsilon}$. We have the following inductive assumptions.

- $|\mathrm{At}(a_k)| \geq 2^{n-k}$ and $|\mathrm{At}(a_e \setminus (a_{e+1} \cup \ldots \cup a_k))| \geq 2^{n-k}$.

- Either $|\mathrm{At}(b_k)| < 2^{n-k}$ or $|\mathrm{At}(b_e \setminus (b_{e+1} \cup \ldots \cup b_k))| < 2^{n-k}$.

There are two cases.

Case 1. Assume that $|\mathrm{At}(b_k)| < 2^{n-k}$ and $|\mathrm{At}(a_k)| \geq 2^{n-k}$. In this case, Spoiler selects a_{k+1} such that $a_{k+1} \subset a_k$, $a_{k+1} \neq \emptyset$, $|\mathrm{At}(a_{k+1})| \geq 2^{n-k-1}$, and $|\mathrm{At}(a_k \setminus a_{k+1})| \geq 2^{n-k-1}$. Note that Duplicator must choose b_{k+1} strictly below b_k. Then either $|\mathrm{At}(b_{k+1})| < 2^{n-k-1}$ or $|\mathrm{At}(b_k \setminus b_{k+1})| < 2^{n-k-1}$

Case 2. Assume that $|\mathrm{At}(b_k)| \geq 2^{n-k}$ and $|\mathrm{At}(a_k)| \geq 2^{n-k}$. In this case, Spoiler selects a_{k+1} such that $a_{k+1} \subset a_e$, $a_{k+1} \neq \emptyset$, $a_{k+1} \cap (a_{e+1} \cup \ldots \cup a_k) = \emptyset$, $|\mathrm{At}(a_{k+1})| \geq 2^{n-k-1}$, and $|\mathrm{At}(a_e \setminus (a_{e+1} \cup \ldots \cup a_{k+1}))| \geq 2^{n-k-1}$. Note that by definition of e, $|\mathrm{At}(b_e)| < 2^{n-k}$ and for each $e+1 \leq i \leq k-1$, $|\mathrm{At}(b_i)| \geq 2^{n-i}$ as otherwise b_k would be below b_i. Hence $|\mathrm{At}(b_k \setminus (b_{e+1} \cup \ldots \cup b_k))| < 2^{n-k}$. Duplicator must choose b_{k+1} strictly below b_e and disjoint with $b_{e+1}, \ldots, b_k$. Therefore, either $|\mathrm{At}(b_{k+1})| < 2^{n-k-1}$ or $|\mathrm{At}(b_e) \setminus \mathrm{At}(b_{e+1} \cup \ldots \cup b_{k+1})| < 2^{n-k-1}$.

After n rounds, by the inductive assumption, it is either $|\mathrm{At}(b_n)| = 0$ or $|\mathrm{At}(b_e \setminus (b_{e+1} \cup \ldots \cup b_n))| = 0$. If the former, then Spoiler wins by selecting $a_{n+1} \subset \mathrm{At}(a_n)$;otherwise, Spoiler wins by selecting $a_{n+1} \subset a_e \setminus (a_{e+1} \cup \ldots \cup a_n)$.

We now prove that the conditions stated in the lemma suffice Duplicator to win the $(n+1)$-round game $G_{n+1}(\mathcal{A}, \mathcal{B})$. Let us assume that the players have produced a k-round

play $(a_1, b_1), (a_2, b_2), ..., (a_k, b_k)$. Our inductive assumptions on this k-round play are the following:

1. The map $a_i \to b_i$ is a partial isomorphism.

2. For each a_i, $i \in \{1, \ldots, k\}$, let $a_i = \cup_\epsilon a_\epsilon$ be as stipulated in Lemma 3.7.1(2). For each a_ϵ, let e be the last round such that $a_\epsilon \subseteq a_e$; if there is no such round, then assume $a_e = \mathrm{At}(I_\epsilon)$. Let d be the last round such that $a_d \subseteq a_\epsilon$; if there is no such round, then assume $a_d = \emptyset$. Let $b_i = \cup_\epsilon b_\epsilon$. The conditions for b_ϵ are the following:

   - $|\mathrm{At}(a_\epsilon \setminus a_d)| \geq 2^{n-i}$ if and only if $|\mathrm{At}(b_\epsilon \setminus a_d)| \geq 2^{n-i}$; $|\mathrm{At}(a_e \setminus a_\epsilon)| \geq 2^{n-i}$ if and only if $|\mathrm{At}(b_e \setminus b_\epsilon)| \geq 2^{n-i}$.
   - If $|\mathrm{At}(a_\epsilon \setminus a_d)| < 2^{n-i}$ then $|\mathrm{At}(b_\epsilon \setminus a_d)| = |\mathrm{At}(a_\epsilon \setminus a_d)|$; If $|\mathrm{At}(a_e \setminus a_\epsilon)| < 2^{n-i}$ then $|\mathrm{At}(b_e \setminus b_\epsilon)| = |\mathrm{At}(a_e \setminus a_\epsilon)|$.

Assume that Spoiler selects an element $a_{k+1} \in A$. Duplicator responds to this move by choosing b_{k+1} as follows. If $a_{k+1} = a_i$ then $b_{k+1} = b_i$. Otherwise, suppose $a_{k+1} = \cup a_\epsilon$ as stipulated in Lemma 3.7.1(2). For each a_ϵ, let d, e be as described in the inductive assumptions. We select each b_ϵ by the following rules.

- If $|\mathrm{At}(a_\epsilon \setminus a_d)| \geq 2^{n-k-1}$ then select b_ϵ such that $|\mathrm{At}(b_\epsilon \setminus a_d)| \geq 2^{n-k-1}$; If $|\mathrm{At}(a_e \setminus a_\epsilon)| \geq 2^{n-k-1}$ then $|\mathrm{At}(b_e \setminus b_\epsilon)| \geq 2^{n-k-1}$.

- If $|\mathrm{At}(a_\epsilon \setminus a_d)| < 2^{n-k-1}$ then select b_ϵ such that $|\mathrm{At}(b_\epsilon \setminus a_d)| = |\mathrm{At}(a_\epsilon \setminus a_d)|$; If $|\mathrm{At}(a_e \setminus a_\epsilon)| < 2^{n-k-1}$ then $|\mathrm{At}(b_e \setminus b_\epsilon)| = |\mathrm{At}(a_e \setminus a_\epsilon)|$.

Finally, Duplicator selects $b_{k+1} \in B$ such that $b_{k+1} = \cup_\epsilon b_\epsilon$.

Note the inductive assumptions guarantee that Duplicator is able to make such a move. It is clear that the inductive assumptions also hold on the $(k+1)$-round play $(a_1, b_1), \ldots, (a_{k+1}, b_{k+1})$. Hence the strategy described must be a winning strategy due to the fact that Duplicator preserves inductive assumption (1) at each round. The lemma is proved. ∎

Theorem 3.7.3 *Fix $n \in \mathbb{N}$. There exists an algorithm that runs in constant time and decides whether Duplicator wins the game $G_{n+1}(\mathcal{A}, \mathcal{B})$ on finite Boolean algebras $\mathcal{A} = (2^{X_A}; \subseteq, \emptyset, X_A, 2^{A_1}, \ldots, 2^{A_s})$ and $\mathcal{B} = (2^{X_B}; \subseteq, \emptyset, X_B, 2^{B_1}, \ldots, 2^{B_s})$. The constant that bounds the running time is $2^s \cdot 2^n$.*

Proof. In order to prove the theorem, we represent the Boolean algebras by listing their atoms in 2^s lists. The i^{th} list lists all atoms with characteristic ϵ_i. To solve game $G_{n+1}(\mathcal{A}, \mathcal{B})$, the algorithm checks the condition in the lemma above by reading the lists. In each list, it reads at most 2^n elements. Therefore the process requires time bounded by $2^s \cdot 2^n$. ∎

Chapter 4

The Complexity of Unary Automatic Structures

This chapter focuses on the class of unary automatic structures, i.e., structures presented by finite automata over the unary alphabet {1}. This class of structures closely resembles finite structures and enjoys some nice algorithmic and logical properties over automatic structures in general. In this chapter, we will study the computational complexity in solving some natural decision problems on these structures. The structures we study are in the signature $\{R\}$ where R is binary relation, i.e., graphs. In particular, we focus on the following classes: graphs of finite degree, equivalence structures, linear orders and trees.

For graphs of finite degree, we provide algorithms of deciding 1) whether there exists an infinite component 2) whether a node belongs to an infinite component 3) whether a node is reachable from another node 4) whether the graph is connected and 5) whether two graphs are isomorphic. The first four algorithms run in polynomial time and are uniform in the automatic presentation of the input graphs, while the fifth algorithm has an elementary time upperbound. For equivalence structures, linear orders and trees, we study the complexity of deciding the membership problem and the isomorphism problem. We also address their state complexities that indicate the sizes of the smallest automata that represent these structures.

4.1 Unary automatic structures

4.1.1 MSO-decidability

The theory of automatic structures can be viewed as an extension of finite model theory in which one studies the interaction between logical definability and computational complexity. In a similar way to the use of finite model theory in reasoning about databases,

automatic structures have been applied to areas where one is interested in the algorithmic properties of infinite structures such as databases and computer-aided verifications [118, 117]. However, this approach has limitations. In particular, since the configuration graph of a Turing machine is automatic (See Example 2.5.8), reachability is undecidable for automatic structure in general. On the other hand, unary automatic structures have decidable monadic second-order theories and hence decidable reachability relation.

Recall that a structure is *unary automatic* if it is automatic over the alphabet $\{1\}$. In this chapter, we use x to denote the word 1^x and thus $\mathbb{N}$ is the language $1^\star$. By [6], a structure is unary automatic if and only if it is FO-interpretable in the structure $\mathcal{U} = (\mathbb{N}; 0, <, s, \{\equiv \mod m\}_{m>1})$, where s is the successor relation and $x \equiv \mod m\, y$ if and only if $x \equiv y \mod m$. Using a monadic second order interpretation of $\mathcal{U}$ in $(\mathbb{N}; s)$ and the decidability of $S1S$[11], one easily get the following result.

Theorem 4.1.1 *The* MSO*-theory of any unary automatic structure is decidable. Furthermore, from a given* MSO*-sentence φ in the signature of the structure, one can construct a Büchi automaton $\mathcal{M}$ such that φ holds if and only if $L(\mathcal{M}) \neq \emptyset$.*

The *reachability* relation on graphs is the transitive closure of the edge relation. Hence, we say a node y is *reachable* from another node x, denoted $\mathrm{Reach}(x, y)$, if there is a path that goes from x to y. From the above theorem, it is not hard to see that the relation Reach is decidable for unary automatic graphs.

The restriction to a unary alphabet is a natural special case of automatic structures because any automatic structure has an isomorphic copy over the binary alphabet (See Prop. 2.5.3). Moreover, even for an intermediate class of automatic structures, e.g., those structures whose domain are encoded as finite strings over $1^\star 2^\star$, reachability is still not decidable as infinite grid can be coded automatically over $1^\star 2^\star$ and counter machines can be coded into the grid. Thus, the class of unary automatic structures is a sensible context where reachability is decidable.

4.1.2 A characterization theorem

We re-state Example 2.4.1 as the following lemma.

Lemma 4.1.2 *A set $L \subseteq \mathbb{N}$ is unary automatic if and only if there are numbers $t, \ell \in \mathbb{N}$ such that $L = L_1 \cup L_2$ with $L_1 \subseteq \{0, 1, \ldots, t-1\}$ and L_2 is a finite union of sets in the form $\{j + i\ell\}_{i \in \mathbb{N}}$ where $t \leq j < t + \ell$.*

We will use the numbers t, ℓ associated with a unary automatic edge relation as parameters in complexity-analysis for classes of unary automatic structures.

Figure 4.1 gives the general shape of a 2-tape unary automaton. We first fix some terminologies. States reachable from the initial state by reading inputs from $(1, 1)^\star$ are

called $(1,1)$-*states*. The set of $(1,1)$-states is a disjoint union of a *tail* and a *loop*. We label the $(1,1)$-states as $q_0, \ldots, q_t, \ldots, q_\ell$ where $q_0, \ldots, q_{t-1}$ form the $(1,1)$-tail and there is a transition from q_ℓ to q_t to close the $(1,1)$-loop. States reachable from a $(1,1)$-state by reading inputs from $(1,\diamond)^\star$ are called $(1,\diamond)$-*states*. The set of $(1,\diamond)$-states reachable from any given q_i consists of a tail and a loop, called the $(1,\diamond)$-*tail* and *loop* from q_i, respectively. The $(\diamond,1)$-*tails* and *loops* are defined similarly. The *tail length* of the automaton is t, the length of its $(1,1)$-tail; the *loop length* is ℓ, the length of its $(1,1)$-loop.

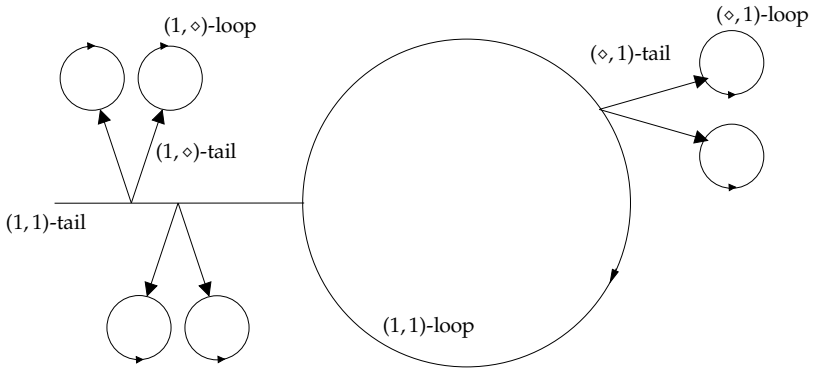


Figure 4.1: General shape of a deterministic 2-tape unary automaton

Khoussainov/Rubin [74] and Blumensath [6] gave a characterization of all unary automatic binary relations on $\mathbb{N}$. Let $F = (V_F; E_F)$ and $D = (F_D; E_D)$ be finite graphs. Let R_1, R_2 be subsets of $V_D \times V_F$, and R_3, R_4 be subsets of $V_F \times V_F$. Similarly, let L_1, L_2 be subsets of $V_F \times V_D$ and L_3, L_4 be subsets of $V_F \times V_F$.

Consider the graph D followed by countably infinitely many copies of F, ordered as $F^0, F^1, F^2, \ldots$. Formally, the node set of F^i is $V_F \times \{i\}$, and we write $b^i = (b, i)$ for $b \in V_F$ and $i \in \mathbb{N}$, the edge set of F^i is denoted by E^i. We define the infinite graph $\mathsf{unwind}(F, D, \bar{R}, \bar{L})$ as follows: Its nodes are $V_D \cup \bigcup_{i\in\mathbb{N}} V_F^i$ and its edge set contains $E_D \cup \bigcup_{i\in\mathbb{N}} E^i$ as well as the following edges, for all $a, b \in V_F$, $d \in V_D$, and $i, j \in \mathbb{N}$:

- (d, b^0) when $(d, b) \in R_1$, and (d, b^{i+1}) when $(d, b) \in R_2$,
- (a^i, b^{i+1}) when $(a, b) \in R_3$, and (a^i, b^{i+2+j}) when $(a, b) \in R_4$,
- (b^0, d) when $(b, d) \in L_1$, and (b^{i+1}, d) when $(b, d) \in L_2$,
- (a^{i+1}, b^i) when $(a, b) \in L_3$, and (a^i, b^{i+2+j}) when $(a, b) \in L_4$.

See Figure 4.2 for an example of an unwinded graph.

Theorem 4.1.3 ([6, 74]) *A graph is unary automatic if and only if it is isomorphic to the graph* $\mathsf{unwind}(F, D, \bar{R}, \bar{L})$ *for some parameters* $F, D, \bar{R}, \bar{L}$.

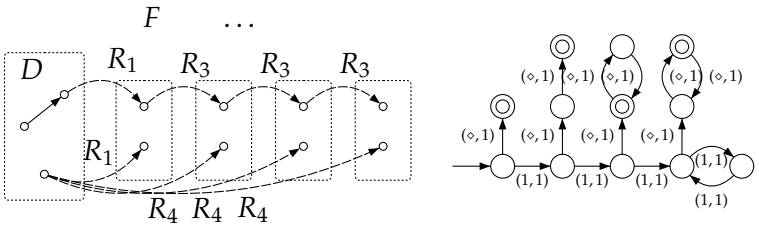


Figure 4.2: An example of unwind$(F, D, \bar{R}, \bar{L})$ and the synchronous 2-tape automaton for its edge relation. If we label $V_F = \{a, b\}$ and $V_D = \{0, 1, 2\}$ then $E_D = \{(0, 1)\}$, $E_F = \emptyset$, $R_1 = \{(1, a), (2, b)\}$, $R_2 = \emptyset$, $R_3 = \{(2, b)\}$, $R_4 = \{(2, b)\}$ and $L_1 = L_2 = L_3 = L_4 = \emptyset$.

4.1.3 Decision problems on unary automatic structures

We investigate from an algorithmic and complexity-theoretical point of view the following problems on unary automatic graphs:

- *Connectivity problem:* Given an automatic graph $\mathcal{G}$, decide whether $\mathcal{G}$ is connected.
- *Reachability problem:* Given an automatic graph $\mathcal{G}$ and two nodes x and y of the graph, decide whether there is a path from x to y.
- *Infinity testing problem:* Given an automatic graph $\mathcal{G}$ and a node x, decide whether the component of $\mathcal{G}$ containing x is infinite.
- *Infinite component problem:* Given an automatic graph $\mathcal{G}$, decide whether $\mathcal{G}$ has an infinite component.

For the class of automatic graphs in general, all of the above problems are undecidable. In fact, one can provide exact bounds on their undecidability: The connectivity problem is Π_2^0-complete; the reachability problem is Σ_1^0-complete; the infinite component problem is Σ_3^0-complete; and the infinity testing problem is Π_2^0-complete [102].

On the other hand, by Theorem 4.1.1 it is not hard to prove that all the problems above are decidable for unary automatic graphs. The focus here is thus put on the computational complexity for deciding these problems. Direct constructions using the MSO-definability yield algorithms with exponential or super-exponential time bounds since one needs to transform MSO-formulas into automata. The question then is whether we can answer the above questions more efficiently than the direct constructions.

Furthermore, we explore algorithms for deciding the following problems:

- *Membership problem:* For a fixed class $\mathfrak{K}$ of structure, decide whether a given unary automatic structure belongs to $\mathfrak{K}$.
- *Isomorphism problem:* Decide whether two given automatic structures are isomorphic.

Alternatively, the membership problem can be stated as follows: Fix a theory T, decide if a given unary automatic structure is a model of T. If T is finitely-axiomatizable, then by Theorem 2.5.11 the problem is clearly decidable. In this case, the focus is put on the computational complexity for deciding this problem.

4.1.4 State complexity of unary automatic structures

The notion of state complexity measures the descriptive complexity of regular languages, context-free grammars, and other classes of languages with finite representations. The state complexity (with respect to automata) of a regular language L is defined to be the size of the smallest automaton with language L. Research into state complexity with respect to automata has been well-established since the 1950s [104, 119, 120]. A key motivation for it is in designing automata for real-time computation where the running time of algorithms depends on the number of states of the automata. In the following definitions, we generalize the notion of state complexity to (unary) automatic structures.

Definition 4.1.4 *The* state complexity *of an (unary) automatic structure $\mathcal{A}$ is the size of the smallest (unary) automaton $\mathcal{M}$ such that $\mathcal{M}$ recognizes an automatic copy of $\mathcal{A}$. We call $\mathcal{M}$ the* optimal (unary) automaton *for $\mathcal{A}$.*

Definition 4.1.5 *Let $\mathfrak{K}$ be a class of (unary) automatic structures such that each member $\mathcal{A}$ of $\mathfrak{K}$ has a finite representation $R_{\mathcal{A}}$ which is independent on the automatic presentation, i.e., for $\mathcal{B} \in \mathfrak{K}$, $\mathcal{A} \cong \mathcal{B}$ if and only if $R_{\mathcal{A}} = R_{\mathcal{B}}$. Let $|R_{\mathcal{A}}|$ denote the size of $R_{\mathcal{A}}$. The* (unary) state complexity *of the class $\mathfrak{K}$ is a function f such that $f(n)$ is the largest (unary) state complexity of all $\mathcal{A} \in \mathfrak{K}$ with $|R_{\mathcal{A}}| \leq n$.*

When measuring the state complexity, we make the following assumptions:

1. Without explicitly mention, all automata are deterministic. Hence, the state complexity that we study here are actually *deterministic state complexity*.

2. All structures are infinite with domain $\mathbb{N}$. Lemma 4.1.6 justifies this assumption. Hence, in this chapter, we assume that an automatic presentation of a structure $(\mathbb{N}; R)$ consists of only the single automaton recognizing R. By an "automaton" recognizing a structure $(\mathbb{N}; R)$, we mean the automaton for R.

3. We assume the sets of $(1,1)$-, $(\diamond, 1)$-, and $(1, \diamond)$- states are pairwise disjoint. Therefore no $(1,1)$-state is also a $(\diamond, 1)$-state etc.

Lemma 4.1.6 *Let $(D; R)$, $D \subset \mathbb{N}$, be a unary automatic binary relation presented by $\mathcal{A}_D$ and $\mathcal{A}_R$. There is a deterministic 2-tape unary automaton $\mathcal{A}_{R'}$, $|\mathcal{A}_{R'}| \leq |\mathcal{A}_R|$, such that $(\mathbb{N}; L(\mathcal{A}_{R'})) \cong (D; R)$.*

Proof. Let t and ℓ be as described in Lemma 4.1.2. We outline the proof in the case when the parameter t associated with L_1 is 0. Let $k_1, k_2, \ldots, k_r$ list all numbers $j \in \{0, \ldots, \ell - 1\}$ such that $\{j + i\ell\}_{i \in \mathbb{N}} \subseteq L_2$ (where L_2 is as defined in Lemma 4.1.2).

Since the binary relation R is defined on the domain D, $\mathcal{A}_R$ must satisfy the following requirements: the $(1,1)$-tail has length $c'\ell$ for some constant c'; the $(1,1)$-loop has length $c\ell$ for some constant c; the lengths of all loops and tails containing accepting states are multiples of ℓ; and, there are no accepting states on any tail or loops off any $(1,1)$-states of the form q_h where $h \notin J$. The isomorphism between D and $\mathbb{N}$ will be given by $i\ell + k_j \mapsto ir + j$. Therefore, define $\mathcal{A}_{R'}$ to have a $(1,1)$-tail of length $c'r$, a $(1,1)$-loop of length cr, and copy the information from the state $i\ell + k_j$ in $\mathcal{A}_R$ to state $i \cdot r + j$ in $\mathcal{A}_{R'}$ (modifying the lengths of $(\diamond, 1)$- and $(1, \diamond)$-tails and loops appropriately). Then, $(\mathbb{N}; L(\mathcal{A}_{R'})) \cong (D; R)$ and since $r \leq \ell$, $\mathcal{A}_{R'}$ has no more states than $\mathcal{A}_R$. ∎

4.2 Unary automatic graphs of finite degree

4.2.1 Characterizations of unary automatic graphs of finite degree

This section studies the algorithmic properties of unary automatic graphs of finite degree. We make use of the following canonical form for 2-tape unary automata:

Definition 4.2.1 *A* one-loop automaton *is an automaton whose transition diagram contains exactly one loop, the* $(1,1)$*-loop, and the lengths of all the tails and loops of the automata equals some number* p*, called the* loop constant*.*

Note that one-loop automata are non-deterministic. If $\mathcal{A}$ is a standard automaton recognizing a binary relation, it has exactly $2p$ (1,1)-states. On each of these states, there is a $(1, \diamond)$-tail and a $(\diamond, 1)$-tail of length exactly p. Therefore if n is the number of states in $\mathcal{A}$, then $n = 4p^2 + 2p$. The following is an easy proposition.

Proposition 4.2.2 *Let* $\mathcal{A}$ *be an* n *state unary automaton recognizing a binary relation* R *on* $1^\star$*. Then* R *has finite degree if and only if there exists a one-loop automaton recognizing* R *with loop constant* $p \leq n$*.*

By the above proposition, we always first convert the input automaton $\mathcal{A}$ into an equivalent one-loop automaton $\mathcal{B}$. In the rest of the paper, we will state all results in terms of the loop constant p (of $\mathcal{B}$) instead of n, the number of states of the input automaton. Since $p \leq n$, for any constant $c > 0$, an $O(p^c)$ algorithm can also be viewed as an $O(n^c)$ algorithm.

Given two unary automatic graphs of finite degree $\mathcal{G}_1 = (V; E_1)$ and $\mathcal{G}_2 = (V; E_2)$ (where we recall the convention that the domain of each graph is $1^\star$), we can form the *union graph* $\mathcal{G}_1 \oplus \mathcal{G}_2 = (V; E_1 \cup E_2)$ and the *intersection graph* $\mathcal{G}_1 \otimes \mathcal{G}_2 = (V; E_1 \cap E_2)$. Automatic graphs of

finite degree are closed under these operations. Indeed, for $i \in \{1,2\}$, let $\mathcal{A}_i$ be a one-loop automaton recognizing E_i with loop constants p_i. The standard construction that builds automata for the union and intersection operations produces a one-loop automaton whose loop constant is $p_1 \cdot p_2$. We now introduce another operation. Consider the new graph $\mathcal{G}'_1 = (V; E'_1)$, where the set E'_1 of edges is defined as follows: a pair $(1^n, 1^m)$ is in E' if and only if $(1^n, 1^m) \notin E$ and $|n-m| \leq p_1$. The relation E'_1 is recognized by the same automaton as E_1, but modified so that all $(\diamond, 1)$-states that are final are declared non-final, and all the $(\diamond, 1)$-states that are non-final are declared final. Thus, we have the following proposition.

Proposition 4.2.3 *If $\mathcal{G}_1$ and $\mathcal{G}_2$ are automatic graphs of finite degree, then so are $\mathcal{G}_1 \oplus \mathcal{G}_2$, $\mathcal{G}_1 \otimes \mathcal{G}_2$ and $\mathcal{G}'_1$.*

We next present an alternative description of the class of unary automatic graphs with finite degree. A *one-counter process*(OCP) is a tuple $\mathbb{O} = (Q, \mathcal{P}, \{Q_p \mid p \in \mathcal{P}\}, \delta_0, \delta_{>0})$ where $\mathcal{P}$ is a countable set of propositions, Q is a finite set of control locations, $Q_p \subseteq Q$ for all $p \in \mathcal{P}$ with $Q_p = \emptyset$ for all but finitely many $p \in P$, $\delta_0 \subseteq Q \times \{0,1\} \times Q$ is a set of zero-transitions, and $\delta_{>0} \subseteq Q \times \{-1,0,1\} \times Q$ is a set of positive-transitions. Recently, verification on one-counter processes has received increasing interests; see for example [33, 107, 112, 32].

Model checking algorithms over a one-counter process $\mathbb{O}$ work on the graph $G(\mathbb{O})$, which is defined as follows: The set of nodes of $G(\mathbb{O})$ is $Q \times \mathbb{N}$ and the edges are

$$\{((a,0),(b,k)) \mid (a,k,b) \in \delta_0, k \in \{0,1\}\} \cup \{((a,i),(b,i+k)) \mid (a,k,b) \in \delta_>, i \in \mathbb{N}, k \in \{-1,0,1\}\}$$

The OCPs can be considered as pushdown automata with just one stack symbol, where the stack serves as a counter. The graph $G(\mathbb{O})$ corresponds to the configuration graph of the pushdown automaton. By Theorem 4.1.3 it is easy to see that the graph $G(\mathbb{O})$ is a unary automatic graph with finite degrees. The following is an easy proposition:

Proposition 4.2.4 *Let G be a graph with finite degree. The following statements are equivalent:*

- *G is unary automatic.*
- *$G \cong \mathsf{unwind}(F, D, \bar{R}, \bar{L})$ for some $F, D, \bar{R}, \bar{L}$ where $R_2 = R_4 = L_2 = L4 = \emptyset$.*
- *$G \cong G(\mathbb{O})$ for some OCP $\mathbb{O}$.*

For convenience, in the rest of the section we always assume the unary automatic graphs are undirected. Therefore, the parameter $\overline{L}$ is symmetric with $\overline{R}$ and is hence omitted. The case when the graphs are directed is treated in a similar manner. In the following we recast Theorem 4.1.3 for graphs of finite degree. Our analysis will show that, in contrast to the general case for automatic graphs, the parameters F, D, and $\overline{R}$ for graphs of finite degree can be extracted in linear time.

Definition 4.2.5 (Unfolding Operation) *Let $\mathcal{D} = (V_\mathcal{D}; E_\mathcal{D})$ and $\mathcal{F} = (V_\mathcal{F}; E_\mathcal{F})$ be finite graphs. Consider the finite sets $\Sigma_{\mathcal{D},\mathcal{F}}$ consisting of all mappings $\eta : V_\mathcal{D} \to P(V_\mathcal{F})$, and $\Sigma_\mathcal{F}$ consisting of all mappings $\sigma : V_\mathcal{F} \to P(V_\mathcal{F})$. Any infinite sequence $\alpha = \eta\sigma_0\sigma_1 \ldots$ where $\eta \in \Sigma_{\mathcal{D},\mathcal{F}}$ and $\sigma_i \in \Sigma_\mathcal{F}$ for each $i \in \mathbb{N}$, defines the infinite graph $\mathcal{G}_\alpha = (V_\alpha; E_\alpha)$ as follows:*

- $V_\alpha = V_\mathcal{D} \cup \{(v, i) \mid v \in V_\mathcal{F}, i \in \omega\}$.
- $E_\alpha = E_\mathcal{D} \cup \{(d, (v, 0)) \mid v \in \eta(d)\} \cup \{((v, i), (v', i)) \mid (v, v') \in E_\mathcal{F}, i \in \omega\} \cup \{((v, i), (v', i+1)) \mid v' \in \sigma_i(v), i \in \omega\}$.

Thus $\mathcal{G}_\alpha$ is obtained by taking $\mathcal{D}$ together with an infinite disjoint union of $\mathcal{F}$ such that edges between $\mathcal{D}$ and the first copy of $\mathcal{F}$ are put according to the mapping η, and edges between successive copies of $\mathcal{F}$ are put according to σ_i.

Figure 4.3 illustrates the general shape of a unary automatic graph of finite degree that is build from $\mathcal{D}$, $\mathcal{F}$, η, and σ^ω, where σ^ω is the infinite word $\sigma\sigma\sigma\cdots$.

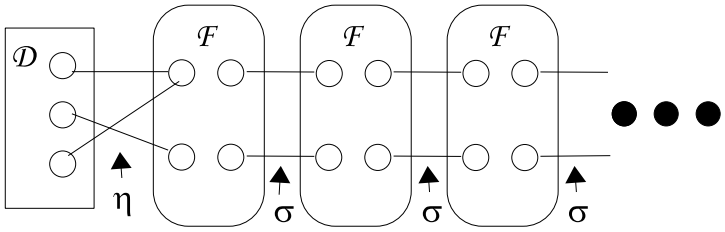


Figure 4.3: Unary automatic graph of finite degree $\mathcal{G}_{\eta\sigma^\omega}$

Theorem 4.2.6 *A graph of finite degree $\mathcal{G} = (V; E)$ possesses a unary automatic presentation if and only if there exist finite graphs $\mathcal{D}, \mathcal{F}$ and mappings $\eta : V_\mathcal{D} \to P(V_\mathcal{F})$ and $\sigma : V_\mathcal{F} \to P(V_\mathcal{F})$ such that $\mathcal{G}$ is isomorphic to $\mathcal{G}_{\eta\sigma^\omega}$.*

Proof. Let $\mathcal{G} = (V; E)$ be a unary automatic graph of finite degree. Let $\mathcal{A}$ be a one-loop automaton recognizing E with loop constant p. We construct the finite graph $\mathcal{D}$ by setting $V_\mathcal{D} = \{q_0, q_1, \ldots, q_{p-1}\}$, where q_0 is the starting state, $q_0, \ldots, q_{p-1}$ are all states on the $(1,1)$-tail such that q_i is reached from q_{i-1} by reading $(1,1)$ for $i > 0$; and for $0 \leq i \leq j < p$, $(q_i, q_j) \in E_\mathcal{D}$ iff there is a final state q_f on the $(\diamond, 1)$-tail out of q_i, and the distance from q_i to q_f is $j - i$. We construct the graph $\mathcal{F}$ similarly by setting $V_\mathcal{F} = \{q'_0, \ldots, q'_{p-1}\}$ where $q'_0, \ldots, q'_{p-1}$ are all states on the $(1,1)$-loop. The edge relation $E_\mathcal{F}$ is defined in a similar way as $E_\mathcal{D}$. The mapping $\eta : V_\mathcal{D} \to P(V_\mathcal{F})$ is defined for any $m, n \in \{0, \ldots, p-1\}$ by putting q'_n in $\eta(q_m)$ if and only if there exists a final state q_f on the $(\diamond, 1)$-tail out of q_m, and the distance from q_m to q_f equals $p + n - m$. The mapping σ is constructed in a similar manner by reading the

$(\diamond, 1)$-tails out of the $(1, 1)$-loop. It is clear from this construction that the graphs $\mathcal{G}$ and $G_{\eta\sigma^\omega}$ are isomorphic.

Conversely, consider the graph $\mathcal{G}_{\eta\sigma^\omega}$ for some $\eta \in \Sigma_{\mathcal{D}}$ and $\sigma \in \Sigma_{\mathcal{F}}$. Assume that $V_{\mathcal{D}} = \{q_0, \ldots, q_{\ell-1}\}$, $V_{\mathcal{F}} = \{q'_0, \ldots, q'_{p-1}\}$. A one-loop automaton $\mathcal{A}$ recognizing the edge relation of $\mathcal{G}_{\eta\sigma^\omega}$ is constructed as follows. The $(1, 1)$-tail of the automaton is formed by $\{q_0, \ldots, q_{\ell-1}\}$ and the $(1, 1)$-loop is formed by $\{q'_0, \ldots, q'_{p-1}\}$, both in natural order. The initial state is q_0. If for some $i < j$, $\{q_i, q_j\} \in E_{\mathcal{D}}$, then put a final state q_f on the $(\diamond, 1)$-tail starting from q_i such that the distance from q_i to q_f is $j - i$. If $q'_j \in \eta(q_i)$, then repeat the process but make the corresponding distance $p + j - i$. The set of edges $E_{\mathcal{F}}$ and mapping σ are treated in a similar manner by putting final states on the $(\diamond, 1)$-tails from the $(1, 1)$-loop.

Again, we see that $\mathcal{A}$ represents a unary automatic graph that is isomorphic to $\mathcal{G}_{\eta\sigma^\omega}$.

∎

The proof of the above theorem also gives us the following corollary.

Corollary 4.2.7 *If $\mathcal{G}$ is a unary automatic graph of finite degree, the parameters $\mathcal{D}$, $\mathcal{F}$, σ and η can be extracted in $O(p^2)$ time, where p is the loop constant of the one-loop automaton representing the graph. Furthermore, $|V_{\mathcal{F}}| = |V_{\mathcal{D}}| = p$.*

4.2.2 Deciding the infinite component problem

Recall the graphs are undirected and a *component* of $\mathcal{G}$ is the transitive closure of a node under the edge relation. The *infinite component problem* asks whether a given graph $\mathcal{G}$ has an infinite component.

Theorem 4.2.8 *The infinite component problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(n^3)$, where p is the loop constant of the one-loop automaton recognizing $\mathcal{G}$.*

By Theorem 4.2.6, let $\mathcal{G} = \mathcal{G}_{\eta\sigma^\omega}$. We observe that it is sufficient to consider the case in which $\mathcal{D} = \emptyset$ (hence $\mathcal{G} = \mathcal{G}_{\sigma^\omega}$) since $\mathcal{G}_{\eta\sigma^\omega}$ has an infinite component if and only if $\mathcal{G}_{\sigma^\omega}$ has one.

Let $\mathcal{F}^i$ be the i^{th} copy of $\mathcal{F}$ in $\mathcal{G}$. Let x^i be the copy of node x in $\mathcal{F}^i$. We construct a finite directed graph $\mathcal{F}^\sigma = (V^\sigma, E^\sigma)$ as follows. Each node in V^σ represents a distinct connected component in $\mathcal{F}$. For simplicity, we assume that $|V^\sigma| = |V_{\mathcal{F}}|$ and hence use x to denote its own component in $\mathcal{F}$. The case in which $|V^\sigma| < |V_{\mathcal{F}}|$ can be treated in a similar way. For $x, y \in V_{\mathcal{F}}$, put $(x, y) \in E^\sigma$ if and only if $y' \in \sigma(x')$ for some x' and y' that are in the same component as x and y, respectively. Constructing $\mathcal{F}^\sigma$ requires finding connected components of $\mathcal{F}$ hence takes time $O(p^2)$. To prove the above theorem, we make essential use of the following definition. See also [49].

Definition 4.2.9 *An oriented walk in a directed graph G is a subgraph $\mathcal{P}$ of G that consists of a sequence of nodes $v_0, ..., v_k$ such that for $i \in \{1, \dots, k\}$, either (v_{i-1}, v_i) or (v_i, v_{i-1}) is an edge in G, and for each $i \in \{0, \dots, k\}$, exactly one of (v_{i-1}, v_i) and (v_i, v_{i-1}) belongs to $\mathcal{P}$. An oriented walk is an oriented cycle if $v_0 = v_k$ and there are no repeated nodes in $v_1, ..., v_k$.*

In an oriented walk $\mathcal{P}$, an edge (v_i, v_{i+1}) is called a *forward edge* and (v_{i+1}, v_i) is called a *backward edge*. The *net length* of $\mathcal{P}$ is the difference between the number of forward edges and backward edges. Note the net length can be negative. The next lemma establishes a connection between oriented cycles in $\mathcal{F}^\sigma$ and infinite components in $\mathcal{G}$.

Lemma 4.2.10 *There is an infinite component in $\mathcal{G}$ if and only if there is an oriented cycle in $\mathcal{F}^\sigma$ such that the net length of the cycle is positive.*

Proof. Suppose there is an oriented cycle $\mathcal{P}$ from x to x in $\mathcal{F}^\sigma$ of net length $m > 0$. For all $i \geq p$, $\mathcal{P}$ defines the path P_i in $\mathcal{G}$ from x^i to x^{i+m} where P_i lies in $\mathcal{F}^{i-p} \cup \cdots \cup \mathcal{F}^{i+p}$. Therefore, for a fixed $i \geq p$, all vertex in the set $\{x^{jm+i} \mid j \in \omega\}$ belong to the same component of $\mathcal{G}$. In particular, this implies that $\mathcal{G}$ contains an infinite component.

Conversely, suppose there is an infinite component D in $\mathcal{G}$. Since $\mathcal{F}$ is finite, there must be some x in $V_\mathcal{F}$ such that there are infinitely many copies of x in D. Let x^i and x^j be two copies of x in D such that $i < j$. Consider a path between x^i and x^j. We can assume that on this path there is at most one copy of any node $y \in V_\mathcal{F}$ apart from x (otherwise, choose x^j to be the copy of x in the path that has this property). By definition of $\mathcal{G}_{\sigma^\omega}$ and $\mathcal{F}^\sigma$, the node x must be on an oriented cycle of $\mathcal{F}^\sigma$ with net length $j - i$. ∎

Proof of Theorem 4.2.8 By the equivalence in Lemma 4.2.10, it suffices to provide an algorithm that decides if $\mathcal{F}^\sigma$ contains an oriented cycle with positive net length. Notice that the existence of an oriented cycle with positive net length is equivalent to the existence of an oriented cycle with negative net length. Therefore, we give an algorithm that finds oriented cycles with non-zero net length.

For each node x in $\mathcal{F}^\sigma$, we search for an oriented cycle of positive net length from x by creating a labeled queue of nodes Q_x which are connected to x.

An important property of this algorithm is that when we are building a queue for node x and are processing z, both $d(z)$ and $d'(z)$ represent net lengths of paths from x to z.

We claim that the algorithm returns **true** if and only if there is an oriented cycle in $\mathcal{F}^\sigma$ with non-zero net length. Suppose the algorithm returns **true**. Then, there is a base node x and a node z such that $d(z) \neq d'(z)$. This means that there is an oriented walk $\mathcal{P}$ from x to z with net length $d(z)$ and there is an oriented walk $\mathcal{P}'$ from x to z with net length $d'(z)$. Consider the oriented walk $\mathcal{P}\overleftarrow{\mathcal{P}'}$, where $\overleftarrow{\mathcal{P}'}$ is the oriented walk $\mathcal{P}'$ in reverse direction. Clearly this is an oriented walk from x to x with net length $d(z) - d'(z) \neq 0$. If there are no repeated nodes in $\mathcal{P}\overleftarrow{\mathcal{P}'}$, then it is the required oriented cycle. Otherwise, let y be a

Algorithm 1 OrientedCycle($\mathcal{F}^\sigma$).

Pick node $x \in V^\sigma$ for which a queue has not been built.
Set the queue Q_x to empty. $d(x) \leftarrow 0$ and put x in Q_x.
unprocessed(x) $\leftarrow$ **true**.
while $\exists y \in Q_x$: unprocessed(y) **do**
    for $z \in \{z \mid (y,z) \in E^\sigma \vee (z,y) \in E^\sigma\}$ **do**
        if $(y,z) \in E^\sigma$ **then** $d'(z) \leftarrow d(y) + 1$. **end if**
        if $(z,y) \in E^\sigma$ **then** $d'(z) \leftarrow d(y) - 1$. **end if**
        if $z \notin Q_x$ **then**
            $d(z) \leftarrow d'(z)$, put z into Q_x, unprocessed(z) $\leftarrow$ **true end if**
        if $z \in Q_x$ **then**
            if $d(z) = d'(z)$ **then** continue;
            if $d(z) \neq d'(z)$ **then** return **true end if**
    end for
    unprocessed(y) $\leftarrow$ **false**
end while
return **false**.

repeated node in $\mathcal{P}\overleftarrow{\mathcal{P}'}$ such that no nodes between the two occurrences of y are repeated. Consider the oriented walk between these two occurrences of y, if it has a non-zero net length, then it is our required oriented cycle; otherwise, we disregard the part between the two occurrences of z and make the oriented walk shorter without altering its net length.

Conversely, suppose there is an oriented cycle $\mathcal{P} = x_0, \ldots, x_m$ of non-zero net length where $x_0 = x_m$. However, we assume for a contradiction that the algorithm returns **false**. Consider how the algorithm acts when we pick x_0 at step (1). For each $i \in \{0, 1, \ldots, m\}$, one can prove the following statements by induction on i.

($\star$) x_i always gets a label $d(x_i)$

($\star\star$) $d(x_i)$ equals the net length of the oriented walk from x_0 to x_i in $\mathcal{P}$.

By the description of the algorithm, x_0 gets the label $d(x_0) = 0$. Suppose the statements holds for x_i, $0 \leq i < m$, then at the next stage, the algorithm labels all nodes in $\{z \mid (z, x_i) \in E^\sigma$ or $(x_i, z) \in E^\sigma\}$. In particular, it calculates $d'(x_{i+1})$. By the inductive hypothesis, $d'(x_{i+1})$ is the net length of the oriented walk from x_0 to x_{i+1} in $\mathcal{P}$. If x_{i+1} has already had a label $d(x_{i+1})$ and $d(x_{i+1}) \neq d'(x_{i+1})$, then the algorithm would return **true**. Therefore $d(x_{i+1}) = d'(x_{i+1})$. By assumption on $\mathcal{P}$, $d(x_m) \neq 0$. However, since $x_0 = x_m$, the induction gives that $d(x_m) = d(x_0) = 0$. This is a contradiction, and thus the above algorithm is correct.

In summary, the following algorithm solves the infinite component problem. Suppose we are given an automaton (with loop constant p) which recognizes the unary automatic graph of finite degree $\mathcal{G}$. Recall that p is also the cardinality of $V_\mathcal{F}$. We first compute $\mathcal{F}^\sigma$,

in time $O(p^2)$. Then we run Algorithm 1 to decide whether $\mathcal{F}^\sigma$ contains an oriented cycle with positive net length. For each node x in $\mathcal{F}^\sigma$, the process runs in time $O(p^2)$. Since $\mathcal{F}^\sigma$ contains p number of nodes, this takes time $O(p^3)$. ∎

4.2.3 Deciding the infinity testing problem

We next turn our attention to the *infinity testing problem* for unary automatic graphs of finite degree. Recall that this problem asks for an algorithm that, given a node v and a graph $\mathcal{G}$, decides if v belongs to an infinite component. We prove the following theorem.

Theorem 4.2.11 *The infinity testing problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(p^3)$, where p is the loop constant of the one-loop automaton $\mathcal{A}$ recognizing $\mathcal{G}$. In particular, when $\mathcal{A}$ is fixed, there is a constant time algorithm that decides the infinity testing problem on $\mathcal{G}$.*

For a fixed input x^i, we have the following lemma.

Lemma 4.2.12 *If x^i is connected to some y^j such that $|j-i| > p$, then x^i is in an infinite component.*

Proof. Suppose such a y^j exists. Take a path P in $\mathcal{G}$ from x^i to y^j. Since p is the cardinality of $V_\mathcal{F}$, there is $z \in V_\mathcal{F}$ such that z^s and z^t appear in P with $s < t$. Therefore all nodes in the set $\{z^{s+(t-s)m} \mid m \in \omega\}$ are in the same component as x^i. ∎

Let $i' = \min\{p, i\}$. To decide whether x^i and y^j are in the same component, we run a breadth first search in $\mathcal{G}$ starting from x^i and going through all nodes in $\mathcal{F}^{i-i'}, \ldots, \mathcal{F}^{i+p}$. See Algorithm 2 as follows:

Algorithm 2 FiniteReach($\mathcal{G}, x^i$).

```
1: i' ← min{p, i}.
2: Set the queue Q to be empty.
3: Put (x, 0) into Q; unprocessed(x, 0) ← true.
4: while ∃(y, d) ∈ Q: unprocessed(y, d) do
5:     for z ∈ {z | (y, z) ∈ E^σ ∨ (z, y) ∈ E^σ} do
6:         if (y, z) ∈ E^σ then d' ← d + 1 end if
7:         if (z, y) ∈ E^σ then d' ← d − 1 end if
8:         if −i' ≤ d' ≤ p and (z, d') ∉ Q then
9:             Put (z, d') into Q; unprocessed(z, d') ← true. end if
10:        end for
11:    unprocessed(y, d) ← false.
12: end while
```

Note that any y^j is reachable from x^i on the graph $\mathcal{G}$ restricted on $\mathcal{F}^{i-i'}, \ldots, \mathcal{F}^{i+p}$ if and only if after running the FiniteReach($\mathcal{G}, x^i$), the pair $(y, j-i)$ is in Q. When running the

algorithm we only use the exact value of the input i when $i < p$ (we set $i' = p - 1$ whenever $i \geq p$), so the running time of FiniteReach$(\mathcal{G}, x^i)$ is bounded by the number of edges in $\mathcal{G}$ restricted to $\mathcal{F}^0, \ldots, \mathcal{F}^{2p}$. Therefore the running time is $O(p^3)$. Let $B = \{y \mid (y, p) \in Q\}$.

Lemma 4.2.13 *Let $x \in V_{\mathcal{F}}$. x^i is in an infinite component if and only if $B \neq \emptyset$.*

Proof. Suppose a node $y \in B$, then there is a path from x^i to y^{i+p}. By Lemma 4.2.12, x^i is in an infinite component. Conversely, if x^i is in an infinite component, then there must be some nodes in $\mathcal{F}^{i+p}$ reachable from x^i. Take a path from x^i to a node y^{i+p} such that y^{i+p} is the first node in $\mathcal{F}^{i+p}$ appearing on this path. Then $y \in B$. ∎

Proof of Theorem 4.2.11 We assume the input node x^i is given by the pair (x, i). The above lemma suggests a simple algorithm to check whether x^i is in an infinite component.

Algorithm 3 InfiniteTest$(\mathcal{G}, x^i)$.

1: Run FiniteReach$(\mathcal{G}, x^i)$, computing the set B while building the queue Q.
2: **for** $y \in B$ **do**
3: **if** $\exists z : (y, z) \in E^\sigma$ **then**
4: Return **true**
5: **end for**. Return **false**

Running FiniteReach$(\mathcal{G}, x^i)$ takes time $O(p^3)$ and checking for edge (y, z) takes $O(p^2)$. The running time is therefore $O(p^3)$. Since x is bounded by p, if $\mathcal{A}$ is fixed, checking whether x^i belongs to an infinite component takes constant time. ∎

4.2.4 Deciding the reachability problem

The reachability problem is studied in [9, 24, 111] on configuration graphs of pushdown automata, i.e., pushdown graphs. It is proved that for a pushdown graph $\mathcal{G}$, given a node v, there is an automaton that recognizes all nodes reachable from v. The number of states in the automaton depends on the input node v. Since unary automatic graphs is a subclass of pushdown graphs, this result implies that there is an algorithm that decides the reachability problem on unary automatic graphs of finite degree. However, there are several issues with this algorithm. The automata constructed by the algorithm are not uniform in v in the sense that different automata are built for different input nodes v. Moreover, the automata are non-deterministic. Hence, the size of the deterministic equivalent automata is exponential in the size of the representation of v.

In this section, we present an alternative algorithm to solve the reachability problem on unary automatic graphs of finite degree uniformly. This new algorithm constructs a deterministic automaton $\mathcal{A}_{\mathsf{Reach}}$ that accepts the relation Reach. Hence, the reachability relation of any unary automatic graph is also unary automatic. The size of $\mathcal{A}_{\mathsf{Reach}}$ only

depends on the number of states of the automaton n, and constructing the automaton requires polynomial time in n. The practical advantage of such a uniform solution is that when $\mathcal{A}_{\mathsf{Reach}}$ is built, deciding whether node v is reachable from u by a path takes only linear time. Our goal in this section is to prove the following theorem.

Theorem 4.2.14 *There exists an algorithm that solves the reachability problem on any unary automatic graph $\mathcal{G}$ of finite degree in time $O(p^4 + |u| + |v|)$ where u, v are two input nodes from the graph $\mathcal{G}$ and p is the loop-constant of the one-loop automaton representing $\mathcal{G}$.*

We restrict to the case when $\mathcal{G} = \mathcal{G}_{\sigma^\omega}$. The proof can be modified slightly to work in the more general case, $\mathcal{G} = \mathcal{G}_{\eta\sigma^\omega}$.

Since, by Theorem 4.2.11, there is an $O(p^3)$-time algorithm to check whether x^i is in a finite component, we can work on the two possible cases separately. We first deal with the case when the input x^i is in a finite component. By Lemma 4.2.12, x^i and y^j are in the same (finite) component if and only if after running $\mathsf{FiniteReach}(\mathcal{G}, x^i)$, the pair $(y, j-i)$ is in the queue Q.

Corollary 4.2.15 *If all components of $\mathcal{G}$ are finite and we represent (x^i, y^j) as $(x^i, y^j, j-i)$, then there is an $O(p^3)$-algorithm that decides whether x^i and y^j are in the same component.* ∎

Now, suppose that x^i is in an infinite component. We start with the following question: given $y \in V_{\mathcal{F}}$, are x^i and y^i in the same component in $\mathcal{G}$? To answer this, we present an algorithm that computes all nodes $y \in V_{\mathcal{F}}$ whose i^{th} copy lies in the same $\mathcal{G}$-component as x^i. The algorithm is the same as $\mathsf{FiniteReach}(\mathcal{G}, x^i)$, except that it does not depend on the input x^i. Hence in Line 1 of Algorithm 2 we set $i' = p$, as opposed to $i' = \min\{p, i\}$.

We use this modified algorithm to define the set $\mathsf{Reach}(x) = \{y \mid (y, 0) \in Q\}$. Intuitively, we can think of the algorithm as a breadth first search through $\mathcal{F}^0 \cup \cdots \cup \mathcal{F}^{2p}$ which originates at x^p. Therefore, $y \in \mathsf{Reach}(x)$ if and only if there exists a path from x^p to y^p in $\mathcal{G}$ restricted to $\mathcal{F}^0 \cup \cdots \cup \mathcal{F}^{2p}$.

Lemma 4.2.16 *Suppose x^i is in an infinite component. The node y^i is in the same component as x^i if and only if y^i is also in an infinite component and $y \in \mathsf{Reach}(x)$.*

Proof. Suppose y^i is in an infinite component and $y \in \mathsf{Reach}(x)$. If $i \geq p$, then the observation above implies that there is a path from x^i to y^i in $\mathcal{F}^{i-p} \cup \cdots \cup \mathcal{F}^{i+p}$. So, it remains to prove that x^i and y^i are in the same component even if $i < p$.

Since $y \in \mathsf{Reach}(x)$, there is a path P in $\mathcal{G}$ from x^p to y^p. Let ℓ be the least number such that $\mathcal{F}^\ell \cap P \neq \emptyset$. If $i \geq p - \ell$, then it is clear that x^i and y^i are in the same component. Thus, suppose that $i < p - \ell$. Let z be such that $z^\ell \in P$. Then P is P_1P_2 where P_1 is a path from x^p to z^ℓ and P_2 is a path from z^ℓ to y^p. Since x^i is in an infinite component, it is easy to see

that x^p is also in an infinite component. There exists an $r > 0$ such that all nodes in the set $\{x^{p+rm} \mid m \in \omega\}$ are in the same component. Likewise, there is an $r' > 0$ such that all nodes in $\{y^{p+r'm} \mid m \in \omega\}$ are in the same component. Consider $x^{p+rr'}$ and $y^{p+rr'}$. Analogous to the path P_1, there is a path P'_1 from $x^{p+rr'}$ to $z^{\ell+rr'}$. Similarly, there is a path P'_2 from $z^{\ell+rr'}$ to $y^{p+rr'}$. We describe another path P' from x^p to y^p as follows. P' first goes from x^p to $x^{p+rr'}$, then goes along $P'_1P'_2$ from $x^{p+rr'}$ to $y^{p+rr'}$ and finally goes to y^p. Notice that the least ℓ' such that $\mathcal{F}_{\ell'} \cap P' \neq \emptyset$ must be larger than ℓ. We can iterate this procedure of lengthening the path between x^p and y^p until $i < p - \ell'$, as is required to reduce to the previous case.

To prove the implication in the other direction, we assume that x^i and y^i are in the same infinite component. Then y^i is, of course, in an infinite component. We want to prove that $y \in \mathsf{Reach}(x)$. Let $i' = \min\{p, i\}$. Suppose there exists a path P in $\mathcal{G}$ from x^i to y^i which stays in $\mathcal{F}^{i-i'} \cup \cdots \cup \mathcal{F}^{i+p}$. Then, indeed, $y \in \mathsf{Reach}(x)$. On the other hand, suppose no such path exists. Since x^i and y^i are in the same component, there is some path P from x^i to y^i. Let $\ell(P)$ be the largest number such that $P \cap \mathcal{F}^{\ell(P)} \neq \emptyset$. Let $\ell'(P)$ be the least number such that $P \cap \mathcal{F}^{\ell'(P)} \neq \emptyset$. We are in one of two cases: $\ell(P) > i + p$ or $\ell'(P) < i - p$. We will prove that if $\ell(P) > i + p$ then there is a path P' from x^i to y^i such that $\ell(P') < \ell(P)$ and $\ell'(P') \geq i - p$. The case in which $\ell'(P) < i - p$ can be handled in a similar manner.

Without loss of generality, we assume $\ell'(P) = i$ since otherwise we can change the input x and make $\ell'(P) = i$. Let z be a node in $\mathcal{F}$ such that $z^{\ell(P)} \in P$. Then P is P_1P_2 where P_1 is a path from x^i to $z^{\ell(p)}$ and P_2 is a path from $z^{\ell(p)}$ to y^i. Since $\ell(P) > i + p$, there must be some s^j and s^{j+k} in P_1 such that $k > 0$. For the same reason, there must be some t^m and t^{m+n} in P_2 such that $n > 0$. Therefore, P contains paths between any consecutive pair of nodes in the sequence $(x^i, s^j, s^{k+j}, z^p, t^{m+n}, t^m, y^i)$. Consider the following sequence of nodes:

$$(x^i, s^j, t^{m+n-k}, t^{m-k}, s^{j-n}, s^{j+k-n}, t^m, y^i).$$

It is easy to check that there exists a path between each pair of consecutive nodes in the sequence. Therefore the above sequence describes a path P' from x^i to y^i. It is easy to see that $\ell(P') = \ell(P) - n$. Also since $\ell'(P) = i$, $\ell'(P') > i - p$. Therefore P' is our desired path. ∎

In the following, we abuse notation by using Reach and σ on subsets of $V_{\mathcal{F}}$. We inductively define a sequence $\mathrm{Cl}_0(x), \mathrm{Cl}_1(x), \ldots$ such that each $\mathrm{Cl}_k(x)$ is a subset of $V_{\mathcal{F}}$. Let $\mathrm{Cl}_0(x) = \mathsf{Reach}(x)$ and For $k > 0$, we define $\mathrm{Cl}_k(x) = \mathsf{Reach}(\sigma(\mathrm{Cl}_{k-1}(x)))$. The following lemma is immediate from this definition.

Lemma 4.2.17 *Suppose x^i is in an infinite component, then x^i and y^j are in the same component if and only if y^j is also in an infinite component and $y \in \mathrm{Cl}_{j-i}(x)$.* ∎

We can use the above lemma to construct a simple-minded algorithm that solves the reachability problem on inputs x^i, y^j.

Algorithm 4 NaiveReach$(\mathcal{G}, x^i, y^j)$.

1: Check whether each of x^i, y^j are in an infinite component (using Alg. 3).
2: **if** exactly one of x^i and y^j is in a finite component **then** return **false**. **end if**
3: **if** both x^i and y^j are in finite components **then**
4: run FiniteReach$(\mathcal{G}, x^i)$; check whether $(y, j-i) \in Q$
5: **end if**
6: **if** both x^i and y^j are in infinite components **then**
7: compute $\mathrm{Cl}_{j-i}(x)$.
8: **if** $y \in \mathrm{Cl}_{j-i}(x)$ **then** return **true**
9: **else** return **false**. **end if**
10: **end if**

We now consider the complexity of this algorithm. The set $\mathrm{Cl}_0(x)$ can be computed in time $O(p^3)$. Given $\mathrm{Cl}_{k-1}(x)$, we can compute $\mathrm{Cl}_k(x)$ in time $O(p^3)$ by computing Reach(y) for any $y \in \sigma(\mathrm{Cl}_{k-1}(x))$. Therefore, the total running time of NaiveReach$(\mathcal{G}, x^i, y^j)$ is $(j-i) \cdot p^3$. We want to replace the multiplication with addition and hence tweak the algorithm.

From Lemma 4.2.13, x^i is in an infinite component in $\mathcal{G}$ if and only if FiniteReach$(\mathcal{G}, x^i)$ finds a node y^{i+p} connecting to x^i. Now, suppose that x^i is in an infinite component. We can use FiniteReach$(\mathcal{G}, x^i)$ to find such a y, and a path from x^i to y^{i+p}. On this path, there must be two nodes z^{i+j}, z^{i+k} with $0 \leq j < k \leq p$. Let $r = k - j$. Note that r can be computed from the algorithm. It is easy to see that all nodes in the set $\{x^{i+mr} \mid m \in \omega\}$ belong to the same component.

Lemma 4.2.18 $\mathrm{Cl}_0(x) = \mathrm{Cl}_r(x)$.

Proof. By definition, $y \in \mathrm{Cl}_0(x)$ if and only if x^p and y^p are in the same component of $\mathcal{G}$. Suppose that there exists a path in $\mathcal{G}$ from x^p to y^p. Then there is a path from x^{p+r} to y^{p+r}. Since x^p and x^{p+r} are in the same component of $\mathcal{G}$, x^p and y^{p+r} are in the same component. Hence $y \in \mathrm{Cl}_r(x)$.

For the reverse inclusion, suppose $y \in \mathrm{Cl}_r(x)$. Then there exists a path from x^p to y^{p+r}. Therefore, x^{p+r} and y^{p+r} are in the same component. Since $r \leq p$, x^p and y^p are in the same component. ∎

Using the above lemma, we define a new algorithm NewReach$(\mathcal{G}, x^i, y^j)$ by replacing the last **if**-statement (Line 6-10) in Alg. 4. See below.

6: **if** x^i and y^j belong to infinite components **then**
7: Compute $\mathrm{Cl}_0(x), \ldots, \mathrm{Cl}_{r-1}(x)$
8: **if** $\exists k < r : y \in \mathrm{Cl}_k(x) \wedge j - i \equiv k \mod r$ **then** return **true**.

9: **else** return **false end if**
10: **end if**

Proof of Theorem 4.2.14. Say input nodes are given as x^i and y^j. By Lemma 4.2.17 and Lemma 4.2.18, the NewReach$(\mathcal{G}, x^i, y^j)$ algorithm returns **true** if and only if x^i and y^j are in the same component. Since $r \leq p$, calculating $\mathrm{Cl}_0(x), \ldots, \mathrm{Cl}_{r-1}(x)$ requires time $O(p^4)$. Therefore the running time of NewReach$(\mathcal{G}, x^i, y^j)$ on input x^i, y^j is $O(i + j + p^4)$. ∎

Notice that, in fact, the algorithm produces a number $k < p$ such that in order to check whether x^i, y^j $(j > i)$ are in the same component, we need to test if $j - i < p$ and if $j - i = k \bmod p$. Therefore if $\mathcal{G}$ is fixed and we compute $\mathrm{Cl}_0(x), \ldots, \mathrm{Cl}_{r_x-1}(x)$ for all x beforehand, then deciding whether two nodes u, v belong to the same component takes linear time. The above proof can also be used to build an automaton that decides reachability uniformly:

Corollary 4.2.19 *Given a unary automatic graph of finite degree $\mathcal{G}$ represented by an automaton with loop constant p, there is a deterministic automaton $\mathcal{A}_{\mathsf{Reach}}$ with at most $2p^4 + p^3$ states that accepts the reachability relation of $\mathcal{G}$. Furthermore, the time required to construct $\mathcal{A}_{\mathsf{Reach}}$ is $O(p^5)$.*

Proof. For all $0 \leq x < p$, $i \in \omega$, let string 1^{ip+x} represent node x^i in $\mathcal{G}$. Suppose $ip + x \leq jp + y$, we construct an automaton $\mathcal{A}_{\mathsf{Reach}}$ that accepts $(1^{ip+x}, 1^{jp+y})$ if and only if x^i and y^j are in the same component in $\mathcal{G}$.

1. $\mathcal{A}_{\mathsf{Reach}}$ has a $(1,1)$-tail of length p^2. Let the states on the tail be $q_0, q_1, \ldots, q_{p^2-1}$, where q_0 is the initial state. These states represent nodes in $\mathcal{F}^0, \mathcal{F}^1, \ldots, \mathcal{F}^{p-1}$.

2. From q_{p^2-1}, there is a $(1,1)$-loop of length p. We call the states on the loop $q'_0, q'_1, \ldots, q'_{p-1}$. These states represent nodes in $\mathcal{F}^p$.

3. For $0 \leq x, i < p$, there is a $(\diamond, 1)$-tail from q_{ip+x} of length $p^2 - x$. We denote the states on this tail by $q^1_{ip+x}, \ldots, q^{p^2-x}_{ip+x}$. These states represent nodes in $\mathcal{F}^i, \mathcal{F}^{i+1}, \ldots, \mathcal{F}^{i+p-1}$.

4. For $0 \leq x, i \leq p$, if x^i is in an infinite component, then there is a $(\diamond, 1)$-loop of length $r \times p$ from $q^{p^2-x}_{ip+x}$. The states on this loop are called $\check{q}^1_{ip+x}, \ldots, \check{q}^{rp}_{ip+x}$. These states represent nodes in $\mathcal{F}^{i+p}, \ldots, \mathcal{F}^{i+p+r-1}$.

5. For $0 \leq x \leq p$, if x^p is in a finite component, then there is a $(\diamond, 1)$-tail from q'_x of length p^2. These states are denoted $\hat{q}^1_x, \ldots, \hat{q}^{p^2}_x$ and represent nodes in $\mathcal{F}_p, \ldots, \mathcal{F}_{2p-1}$.

6. If x^p is in an infinite component, from q'_x, there is a $(\diamond, 1)$-loop of length $r \times p$. We write these states as $\tilde{q}^1_x, \ldots, \tilde{q}^{rp}_x$.

The final (accepting) states of $\mathcal{A}_{\mathsf{Reach}}$ are defined as follows:

1. States $q_0, \ldots, q_{p^2-1}, q'_0, \ldots, q_{p-1}$ are final.
2. For $i < p$, if x^i is in a finite component, run the algorithm $\mathsf{FiniteReach}(\mathcal{G}, x^i)$ and declare state q_{ip+x}^{jp+y-x} final if $(y, j) \in Q$.
3. For $i < p$, if x^i is in an infinite component, compute $\mathrm{Cl}_0(x), \ldots, \mathrm{Cl}_{r-1}(x)$.
   (a) Make state q_{ip+x}^{jp+y-x} final if y^{i+j} is in an infinite component and $y \in \mathrm{Cl}_j(x)$.
   (b) Make state $\check{q}_{ip+x}^{jp+y-x}$ final if $y \in \mathrm{Cl}_j(x)$
4. If x^p is in a finite component, run the algorithm $\mathsf{FiniteReach}(\mathcal{G}, x^p)$ and make state $\hat{q}_x^{jp+y-x}$ final if $(y, j) \in Q$.
5. If x^p is in an infinite component, compute $\mathrm{Cl}_0(x), \ldots, \mathrm{Cl}_{r-1}(x)$. Declare state $\tilde{q}_x^{jp+y-x}$ final if $y \in \mathrm{Cl}_j(x)$.

One can show that $\mathcal{A}_{\mathsf{Reach}}$ is the desired automaton. To compute the complexity of building $\mathcal{A}_{\mathsf{Reach}}$, we summarize the computation involved.

1. For all x^i in $\mathcal{F}^0 \cup \cdots \cup \mathcal{F}^p$, decide whether x^i is in a finite component. This takes time $O(p^5)$ by Theorem 4.2.11.
2. For all x^i in $\mathcal{F}^0 \cup \cdots \cup \mathcal{F}^p$ such that x^i is in a finite component, run $\mathsf{FiniteReach}(\mathcal{G}, x^i)$. This takes time $O(p^5)$ by Corollary 4.2.15.
3. For all $x \in V_{\mathcal{F}}$ such that x^p is in an infinite component, compute the sets $\mathrm{Cl}_0(x)$, $\ldots, \mathrm{Cl}_{r-1}(x)$. This requires time $O(p^5)$ by Theorem 4.2.14.

Therefore the running time required to construct $\mathcal{A}_{\mathsf{Reach}}$ is $O(p^5)$. ∎

4.2.5 Deciding the connectivity problem

We now present a solution to the *connectivity problem* on unary automatic graphs of finite degree. Recall a graph is *connected* if there is a path between any pair of nodes. The construction of $\mathcal{A}_{\mathsf{Reach}}$ from the last section suggests an immediate solution to the connectivity problem.

The above algorithm takes time $O(p^5)$. Note that $\mathcal{A}_{\mathsf{Reach}}$ provides a uniform solution to the reachability problem on $\mathcal{G}$. Given the "regularity" of the class of infinite graphs we are studying, it is reasonable to believe there is a more intuitive algorithm that solves the connectivity problem. It turns out that this is the case.

Algorithm 5 NaiveConnect($\mathcal{G}$).

1: Construct the automaton $\mathcal{A}_{\mathsf{Reach}}$.
2: **if** all states in $\mathcal{A}_{\mathsf{Reach}}$ are final states **then** return **true**
3: **else** return **false end if**

Theorem 4.2.20 *The connectivity problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(p^3)$, where p is the loop constant of the automaton recognizing $\mathcal{G}$.*

Observe that if $\mathcal{G}$ does not contain an infinite component, then $\mathcal{G}$ is not connected. Therefore we suppose $\mathcal{G}$ contains an infinite component C.

Lemma 4.2.21 *For all $i \in \mathbb{N}$, there is a node in $\mathcal{F}^i$ belonging to C.*

Proof. Since C is infinite, there is a node x^i and $s > 0$ such that all nodes in $\{x^{i+ms} \mid m \in \omega\}$ belong to C and i is the least such number. By minimality, $i < s$. Take a walk along the path from x^{i+s} to x^i. Let y^s be the first node in $\mathcal{F}^s$ that appears on this path. It is easy to see that y^0 must also be in C. Therefore, C has a non-empty intersection with each copy of $\mathcal{F}$ in $\mathcal{G}$. ∎

Pick an arbitrary $x \in V_{\mathcal{F}}$ and run FiniteReach($\mathcal{G}, x^0$) on x^0 to compute the queue Q. Set $R = \{y \in V_{\mathcal{F}} \mid (y, 0) \in Q\}$.

Lemma 4.2.22 *Suppose $\mathcal{G}$ contains an infinite component, then $\mathcal{G}$ is connected if and only if $R = V_{\mathcal{F}}$.*

Proof. Suppose there is a node $y \in V_{\mathcal{F}} \setminus R$. Then there is no path in $\mathcal{G}$ between x^0 to y^0. Otherwise, we can shorten the path from x^0 to y^0 using an argument similar to the proof of Lemma 4.2.16, and show the existence of a path between x^0 to y^0 in the subgraph restricted on $\mathcal{F}^0, \ldots, \mathcal{F}^p$. Therefore $\mathcal{G}$ is not connected. Conversely, if $R = V_{\mathcal{F}}$, then every set of the form $\{y \in V_{\mathcal{F}} \mid (y, i) \in Q\}$ for $i \geq 0$ equals $V_{\mathcal{F}}$. By Lemma 4.2.21, all nodes are in the same component. ∎

Proof of Theorem 4.2.20. By the above lemma, the following algorithm decides the connectivity problem on G:

Solving the infinite component problem takes time $O(p^3)$ by Theorem 4.2.8. Running FiniteReach($\mathcal{G}, x^0$) also takes time $O(p^3)$. Therefore the Connectivity($\mathcal{G}$) algorithm takes time $O(p^3)$. ∎

Algorithm 6 Connectivity($\mathcal{G}$)

Use the algorithm proposed by Theorem 4.2.8 to decide whether there is an infinite component in $\mathcal{G}$.
Pick an arbitrary $x \in V_{\mathcal{F}}$, run FiniteReach($\mathcal{G}, x^0$) to compute the queue Q.
Let $C = \{y \mid (y, 0) \in Q\}$.
if $C = V_{\mathcal{F}}$ **then** return **true**. **else** return **false**. **end if**

4.2.6 Deciding the isomorphism problem

We next prove the decidability of the isomorphism problem for unary automatic graphs of finite degree. Instead of the combinatorial approach adopted to solve the other decision problems above, we decide the isomorphism problem using MSO-decidablity (Theorem 4.1.1).

Theorem 4.2.23 *The isomorphism problem for unary automatic graphs of finite degree is decidable in elementary time.*

By Theorem 4.2.6, any infinite component in $\mathcal{G}$ has nonempty intersection with almost all $\mathcal{F}_i$'s. We say a component C *starts in* $\mathcal{F}_i$ if $C \cap \mathcal{F}_i \neq \emptyset$ and for $j < i$, $C \cap \mathcal{F}_j = \emptyset$.

Lemma 4.2.24 *For any finite graph $\mathcal{H}$, there are infinitely many components in $\mathcal{G}$ isomorphic to $\mathcal{H}$ if and only if $\mathcal{H} \cong C$ for some component C starting in $\mathcal{F}_p$.*

Proof. Suppose $C \cong \mathcal{H}$ and C starts in $\mathcal{F}_p$. For each j, $\{x^{i+j} \mid x^i \in C\}$ is a finite component isomorphic to $\mathcal{H}$. On the other hand, if $\mathcal{H}$ is isomorphic to infinitely many components in $\mathcal{G}$, it is isomorphic to some C' that starts in $\mathcal{F}_k$ for $k \geq p$. Then $\{x^{i+p-k} \mid x^i \in C'\}$ is the desired component. ∎

Let $\mathcal{G}_{\mathsf{Fin}}$ be the subgraph of $\mathcal{G}$ containing only its finite components. By Theorem 4.2.6 and Lemma 4.2.24, if C is any finite component of $\mathcal{G}$ then either $C \cap \mathcal{F}_j \neq \emptyset$ for some $j < \ell$ or C has infinitely many isomorphic copies in $\mathcal{G}$. By Lemma 4.2.24, in $\mathcal{G}_{\mathsf{Fin}}$ there are only finitely many isomorphism classes of finite components of $\mathcal{G}$, and we can decide which of these classes correspond to infinitely many components in $\mathcal{G}$. Since finite graph isomorphism is decidable, given two graphs $\mathcal{G}, \mathcal{G}'$ we can decide whether $\mathcal{G}_{\mathsf{Fin}} \cong \mathcal{G}'_{\mathsf{Fin}}$.

Since $\mathcal{G}$ contains only finitely many infinite components, it remains to prove that, given two infinite components of unary automatic graphs, we can check whether they are isomorphic. Note that each infinite component of $\mathcal{G}$ is recognizable by a unary automaton using operations on the automaton $\mathcal{A}_{\mathsf{Reach}}$ described in Corollary 4.2.19. Therefore, it suffices to prove that we can decide whether two infinite connected unary automatic graphs are isomorphic.

To prove Theorem 4.2.23, we will give an (MSO + $\exists^\infty$)-definition of the isomorphism type of a connected graph $\mathcal{G} = (\mathbb{N}; E)$ and then use the decidability of the (MSO+$\exists^\infty$)-theory of unary automatic structures. We first define auxiliary MSO-formulae.

For a fixed set S and $k \in \mathbb{N}$, let $\mathsf{Partition}_k^S(P_1, \ldots, P_k)$ be the formula

$$\left(\bigwedge_{i=1}^{k} \exists^\infty x : \ x \in P_i \right) \wedge \left(\bigwedge_{1 \leq i \neq j \leq k} P_i \cap P_j = \emptyset \right) \wedge \left(S = \bigcup_{i=1}^{k} P_i \right)$$

In other words: S is partitioned into k infinite subsets $P_1, \ldots, P_k$.

For a finite graph $\mathcal{F} = (\{v_1, \ldots, v_k\}; E_\mathcal{F})$, $\mathsf{Type}^\mathcal{F}(X, Y_1, \ldots, Y_k)$ is

$$\exists x_1, \ldots, x_k : \bigwedge_{i=1}^{k} (x_i \in X \cap Y_i \wedge \forall y : y \in X \cap Y_i \rightarrow x_i = y) \wedge$$
$$X = \{x_1, \ldots, x_k\} \wedge \bigwedge_{i,j=1}^{k} E(x_i, x_j) \leftrightarrow E_\mathcal{F}(v_i, v_j)$$

In other words: the set X contains exactly one node x_i from each of the sets Y_i, $1 \leq i \leq k$, and the mapping $x_i \mapsto v_i$ is an isomorphism between the induced graph on X and $\mathcal{F}$. Note that this formula implies that the graph $\mathcal{F}$ has size exactly k.

For a finite graph $\mathcal{F}$ of size k and a mapping $\sigma : V_\mathcal{F} \rightarrow P(V_\mathcal{F})$, let $\mathcal{F}_{\times 3}$ be the finite subgraph of $\mathcal{F}_{\sigma^\omega}$ induced on the first three copies of $\mathcal{F}$ and let $V_{\mathcal{F}_{\times 3}}$ denote its set of nodes. Label $V_{\mathcal{F}_{\times 3}}$ by $v_1, \ldots, v_{3k}$ where $\{v_{ik+1}, \ldots, v_{(i+1)k}\}$ belong to the i^{th} copy of $\mathcal{F}$ for $i \in \{0, 1, 2\}$ and for each $j \in \{1, \ldots, k\}$, the nodes v_j, v_{k+j}, v_{2k+j} all correspond to the same node in $\mathcal{F}$. Define the formula $\mathsf{Succ}_\sigma^\mathcal{F}(X, Y, Z_1, \ldots, Z_{3k})$ to be

$$\mathsf{Type}^\mathcal{F}(X, Z_1, \ldots, Z_{3k}) \wedge \mathsf{Type}^\mathcal{F}(Y, Z_1, \ldots, Z_{3k}) \wedge (X \cap Y = \emptyset)$$
$$\wedge \bigwedge_{(i,j): v_j \in \sigma(v_i)} \forall x \in X \cap Z_{2k+i} \forall y \in Z_j : E(x, y)$$
$$\wedge \bigwedge_{(i,j): v_j \notin \sigma(v_i)} \forall x \in X \cap Z_{2k+i} \forall y \in Z_j : \neg E(x, y)$$
$$\wedge \bigwedge_{(i,j) \notin \{2k+1, \ldots, 3k\} \times \{1, \ldots, k\}} \forall x \in X \cap Z_i \forall y \in Z_j : \neg E(x, y).$$

In other words: the induced graphs on X and Y form two disjoint copies of $\mathcal{F}$ (in the sense as described for $\mathsf{Type}^\mathcal{F}(X, Y_1, \ldots, Y_k)$) which has size $3k$, and these two copies of $\mathcal{F}$ are connected via edges between $X \restriction Z_{2k+1} \cup \ldots \cup Z_{3k}$ and $Y \restriction Z_1 \cup \ldots \cup Z_k$ that respect the mapping σ.

We are now ready to prove the theorem.

Proof of Theorem 4.2.23. $V_{\mathcal{F}} = \{v_0, \ldots, v_k\}$ and recall the definition of $\mathcal{F}_{\times 3}$ above.

We define $\varphi_{\mathcal{G}}$ as $\exists P_1 \cdots \exists P_{3k} : \psi_{\mathcal{G}}(P_1, \ldots, P_{3k})$, where $\psi_{\mathcal{G}}(\overline{Z})$ is the conjunction of the following formulas:

1. $\mathsf{Partition}_{3k}^{\mathbb{N}}(\overline{Z})$

2. $\forall x \exists X : x \in X \wedge \mathsf{Type}^{\mathcal{F}_{\times 3}}(X, \overline{Z})$

3. $\forall X : \mathsf{Type}^{\mathcal{F}_{\times 3}}(X, \overline{Z}) \rightarrow \exists^{=1} Y : \mathsf{Succ}_{\sigma}^{\mathcal{F}}(X, Y, \overline{Z})$

4. $\exists X : \mathsf{Type}^{\mathcal{F}_{\times 3}}(X, \overline{Z}) \wedge \forall Y : (\mathsf{Type}^{\mathcal{F}_{\times 3}}(Y, \overline{Z}) \wedge X \cap Y = \emptyset)$
$\rightarrow [\neg \mathsf{Succ}_{\sigma}^{\mathcal{F}}(X, Y, \overline{Z}) \wedge \exists^{=1} W : \mathsf{Succ}_{\sigma}^{\mathcal{F}}(W, Y, \overline{Z})]]]$

Claim. If $\mathcal{H}$ is an infinite connected graph, $\mathcal{H} \models \varphi_{\mathcal{G}}$ if and only if $\mathcal{H} \cong \mathcal{G}$.

Proof of claim. If $\mathcal{H} \cong \mathcal{G}$ then clearly $\mathcal{H} \models \varphi_{\mathcal{G}}$. On the other hand, suppose $\mathcal{H} \models \varphi_{\mathcal{G}}$. Then $\mathcal{H}$ can be partitioned into $3k$ sets $P_1, \ldots, P_{3k}$. Take a subgraph $\mathcal{M}$ of $3k$ nodes in $\mathcal{H}$. We say that $\mathcal{M}$ is a $\mathcal{F}_{\times 3}$-*type* if $\mathcal{M}$ intersects with each P_i at exactly one node, and if we let v_i be the unique node in $\mathcal{M} \cap P_i$, then the three sets of nodes $\{v_1, \ldots, v_k\}$, $\{v_{k+1}, \ldots, v_{2k}\}$, $\{v_{2k+1}, \ldots, v_{3k}\}$ respectively form three copies of $\mathcal{F}$, with v_i, v_{k+i}, v_{2k+i} corresponding to the same node in $\mathcal{F}$. Also, the edge relation between these three copies of $\mathcal{F}$ respects the mapping σ.

Since $\mathcal{H} \models \varphi_{\mathcal{G}}$, each node v in $\mathcal{H}$ belongs a unique subgraph that is a $\mathcal{F}_{\times 3}$-type; and, for each $\mathcal{F}_{\times 3}$-type $\mathcal{M}$, there is a unique $\mathcal{F}_{\times 3}$-type $\mathcal{N}$ that is a *successor* of $\mathcal{M}$, i.e., all edges between $\mathcal{M}$ and $\mathcal{N}$ are from the last copy of $\mathcal{F}$ in $\mathcal{M}$ to the first copy of $\mathcal{F}$ in $\mathcal{N}$ such that they respect the mapping σ. Lastly there exists a unique $\mathcal{F}_{\times 3}$-type $\mathcal{M}_0$ which is not the successor of any other $\mathcal{F}_{\times 3}$-types and any other $\mathcal{F}_{\times 3}$-type is the successor of a unique $\mathcal{F}_{\times 3}$-type. Note that the successor relation between the $\mathcal{F}_{\times 3}$-types resembles the unfolding operation on finite graphs.

Therefore, to set up an isomorphism from $\mathcal{H}$ to $\mathcal{G}$, we only need to map $\mathcal{M}_0$ isomorphically to the first 3 copies of $\mathcal{F}$ in $\mathcal{G}$, and then map the other nodes according to the successor relation and mapping σ.

By Theorem 4.1.1, satisfiability of any MSO-sentence is decidable for unary automatic graphs. Therefore the isomorphism problem for unary automatic graphs of finite degree is decidable. Since $\varphi_{\mathcal{G}}$ contains a fixed number of nested quantifiers (regardless of the size of the automaton presenting it), the decision procedure is elementary in terms of the size of the input automaton. ∎

4.3 Unary automatic linear orders

4.3.1 A characterization theorem

Recall that a linear order $\mathcal{L}$ is a total partial order. Note that linear orders, when considered as graphs, may not have finite degree. Therefore, in this and the subsequent sections we do not assume the input automata are one-loop automata. The following lemma is immediate.

Proposition 4.3.1 *The membership problem for automatic linear orders is decidable in time $O(n^3)$ where n is the number of states in the automata recognizing the input linear orders.*

Proof. Let $(\mathbb{N}; R)$ be a unary automatic structure where R is binary. Suppose $\mathcal{A}_R$ (n states) is a deterministic finite automata recognizing R. To check whether R is reflexive, we construct an automaton for $\{x \mid (x,x) \in R\}$ and check whether $\{x \mid (x,x) \in R\} = \mathbb{N}$. This takes time $O(n)$. To decide whether R is antisymmetric, we construct an automaton for $S = \{(x,y) \mid x \neq y\}$ and determine whether $R \cap S = \emptyset$. This takes time $O(n^2)$. To decide whether R is total, we construct an automaton for $S_1 = \{(y,x) \mid (x,y) \in R\}$ and decide whether $R \cup S_1 = \mathbb{N}^2$. This takes time $O(n^2)$. Finally, to settle whether R is transitive, we construct the automaton $\{(x,y,z) \mid R(x,y) \wedge R(y,z) \wedge \neg R(x,z)\}$ and check whether its language is empty. This takes time $O(n^3)$. ∎

Recall that ω, ω^*, ζ and $\mathbf{n}$ denote respectively the linear order of the natural numbers, negative numbers, integers and finite linear order of length n. The following theorem was proved by Blumensath [6] and Khoussainov/Rubin [74] and characterizes unary automatic linear orders.

Theorem 4.3.2 *A linear order is unary automatic if and only if it is isomorphic to a finite sum of linear orders of type ω, ω^* or $\mathbf{n}$ where $n \in \mathbb{N}$.*

By Theorem 4.3.2, $\mathcal{L}$ is a linear order of the form $\alpha_1 + \cdots + \alpha_k$ where each α_i, $0 \leq i \leq k$, is one of the linear order in $\{\omega, \omega^*, \zeta, (\mathbf{n})_{n\in\omega}\}$. Thus we denote $\mathcal{L}$ by the *canonical word* $\alpha_{\mathcal{L}} = \alpha_1 \cdots \alpha_k \in \{\omega, \omega^*, (\mathbf{n})_{n\in\omega}\}^\star$. Without loss of generality, we assume further that $\alpha_{\mathcal{L}}$ has no substring of the form $\omega^*\omega$, $\mathbf{n}\omega$, $\omega^*\mathbf{n}$ or $\mathbf{n_1 n_2}$ for $n, n_1, n_2 \in \mathbb{N}$. The following lemma is immediate.

Lemma 4.3.3 *Two unary automatic linear orders $\mathcal{L}_1, \mathcal{L}_2$ are isomorphic if and only if $\alpha_{\mathcal{L}_1} = \alpha_{\mathcal{L}_2}$.*

Corollary 4.3.4 *The isomorphism problem for unary automatic linear orders is decidable.*

Proof. Let $\mathcal{L} = (\mathbb{N}; \leq_{\mathcal{L}})$ be a unary automatic linear order. We will define a $(\mathsf{FO} + \exists^\infty)$-formula $\varphi_{\mathcal{L}}$ such that a linear order $\mathcal{L}_1$ has canonical word $\alpha_{\mathcal{L}_1} = \alpha_{\mathcal{L}}$ if and only if $\mathcal{L}_1 \models$

$\varphi_{\mathcal{L}}$. By Lemma 4.3.3 and Theorem 2.5.11 this proves the decidability of the isomorphism problem.

To define $\varphi_{\mathcal{L}}$, we define the following auxiliary formulas. For $x, y \in \mathbb{N}$, let $\mathsf{FinDis}(x, y)$ be

$$x <_L y \wedge \neg \exists^{\infty} z : x <_L z \wedge z <_L y.$$

For $x \in \omega$, let $\mathsf{In}^{\omega}(x)$ be the formula

$$[\exists^{\infty} y : x <_L y \wedge \mathsf{FinDis}(x, y)] \wedge [\forall z <_L x : \neg \mathsf{FinDis}(z, x)]$$

Let $\mathsf{In}^{\omega^*}(x)$ be the formula

$$[\exists^{\infty} y : y <_L x \wedge \mathsf{FinDis}(y, x)] \wedge [\forall z >_L x : \neg \mathsf{FinDis}(x, z)]$$

Let $\mathsf{In}^{\zeta}(x)$ be the formula

$$[\exists^{\infty} y : x <_L y \wedge \mathsf{FinDis}(x, y)] \wedge [\exists^{\infty} z : z <_L x \wedge \mathsf{FinDis}(z, x)]$$

For any $n \in \omega$, let $\mathsf{In}^{n}(x)$ be the formula

$$\exists y_1, \ldots, y_{n-1} : x <_L y_1 \wedge \bigwedge_{i=1}^{n-2} (y_i <_L y_{i+1}) \wedge \forall z : \neg \mathsf{FinDis}(z, x) \wedge$$
$$\forall z : \mathsf{FinDis}(x, z) \rightarrow z = x \vee \bigvee_{i=1}^{n-1} (z = y_i)$$

Recall that $\alpha_{\mathcal{L}} = \alpha_1 \cdots \alpha_k$ is the canonical word of $\mathcal{L}$. Hence, we define $\varphi_{\mathcal{L}}$ as follows

$$\exists x_0, \ldots, x_{k-1} : \bigwedge_{i=0}^{k-2} (x_i <_L x_{i+1}) \wedge \bigwedge_{i=0}^{k-1} \mathsf{In}^{\alpha_i}(x_i) \wedge$$
$$\forall y : \bigvee_{i=0}^{k-1} (\mathsf{FinDis}(x_i, y) \vee \mathsf{FinDis}(y, x_i))$$

∎

The sentence $\varphi_{\mathcal{L}}$ contains three alternations of quantifiers. To decide whether automatic linear orders $\mathcal{L}, \mathcal{L}'$ are isomorphic, we check whether $\mathcal{L}' \vDash \varphi_{\mathcal{L}}$ (by Theorem 2.5.11). An exponential runtime blow-up occurs for each alternation of quantifiers in $\varphi_{\mathcal{L}}$ [70] and hence the algorithm takes triply exponential time in the size of the input automata. Furthermore, the algorithm is non-uniform as it requires that the formula $\varphi_{\mathcal{L}}$ (and hence the canonical word $\alpha_{\mathcal{L}}$) is known beforehand. We now provide an alternative algorithm which

significantly improves the time complexity and is uniform in the input automata.

4.3.2 An efficient solution to the isomorphism problem

Theorem 4.3.5 *The isomorphism problem for unary automatic linear orders is decidable in time quadratic in the sizes of the input automata.*

Suppose $\mathcal{A} = (Q, \Delta, I, F)$ is a unary automaton that represents a linear order $\mathcal{L} = (\mathbb{N}; \leq_{\mathcal{L}})$. We use the notation from Section 4.1.2: the parameters t, ℓ are resp. the lengths of the $(1,1)$-tail and $(1,1)$-loop in $\mathcal{A}$. For $i \in \{0, \ldots, t-1\}$, let W_i be the singleton $\{i\}$. For $i \in \{t, \ldots, t+\ell-1\}$, let W_i be the set of numbers $\{t + i + j\ell \mid j \in \mathbb{N}\}$. Note that

$$\mathbb{N} = \bigcup_{i=0}^{t+\ell-1} W_i. \tag{4.1}$$

In the following, we will decompose the unary automatic linear orders as a sequence of copies of $\omega, \omega^*, \mathbf{n}$ (with no ζ).

Lemma 4.3.6 *For any $t \leq j < t + \ell$, we have the following:*

1. *If $\Delta(q_j, (\diamond, 1)^\ell) \in F$, the set W_j forms an increasing chain $j <_{\mathcal{L}} j + \ell <_{\mathcal{L}} j + 2\ell <_{\mathcal{L}} \cdots$*
2. *If $\Delta(q_j, (\diamond, 1)^\ell) \notin F$, the set W_j forms an decreasing chain $j >_{\mathcal{L}} j + \ell >_{\mathcal{L}} j + 2\ell >_{\mathcal{L}} \cdots$*
3. *For any $i \in \mathbb{N}$, there are only finitely many elements between $j + i\ell$ and $j + (i+1)\ell$.*

Proof. If $\Delta(q_j, (\diamond, 1)^\ell)$ is an accepting state then $j + i\ell <_{\mathcal{L}} j + (i+1)\ell$ for all i so $(j + i\ell)_{i \in \mathbb{N}}$ is an increasing chain in $\mathcal{L}$. Otherwise, totality of $\mathcal{L}$ implies that $\Delta(q_j, (1, \diamond)^\ell) \in F$ hence $(j + i\ell)_{i \in \mathbb{N}}$ is a decreasing chain in $\mathcal{L}$. This proves (1)(2) in the lemma.

For (3), we only prove the case when $(j + i\ell)_{i \in \mathbb{N}}$ forms an increasing chain. The other case can be proved in a similar way. Let t_1, t_2 be the lengths of the $(\diamond, 1)$- and $(1, \diamond)$- tails off q_j; let ℓ_1, ℓ_2 be the lengths of the $(\diamond, 1)$- and $(1, \diamond)$- loops off q_j. We consider two cases. First, suppose $\ell_1 = \ell_2 = 1$. Since $(j + i\ell)_{i \in \mathbb{N}}$ is increasing, $\Delta(q_j, (\diamond, 1)^{c\ell}) \in F$ for all c. Hence, $\Delta(q_j, (\diamond, 1)^{t_1}) \in F$. Similarly, $\Delta(q_j, (\diamond, 1)^{t_2}) \notin F$. Therefore, each $x > j + (i+1)\ell + \max\{t_1, t_2\}$ satisfies $x >_L j + (i+1)\ell$ and there are only finitely many elements $<_L$-below $j + (i+1)\ell$. This leaves only finitely many possible elements $<_L$-between $j + i\ell$ and $j + (i+1)\ell$.

On the other hand, suppose $\ell_1 \ell_2 > 1$. Let $k = \max\{t_1, t_2\} + \ell$. Suppose there is $i \geq 0$ and $r = j + i\ell + k + s$, $s \geq 0$ such that

$$j + i\ell <_L r <_L j + (i+1)\ell.$$

The first inequality is equivalent to $\Delta(q_j, (\diamond, 1)^{k+s}) \in F$ and hence for any c, $j + i\ell + c\ell <_L r + c\ell$. The second inequality implies that $\Delta(q_j, (1, \diamond)^{k+s-\ell}) \in F$ and is in the $(1, \diamond)$-loop off q_j. So, for any $c' \geq 0$, $r + c'\ell_2 <_L j + (i+1)\ell$. Therefore,

$$j + i\ell + (\ell_1\ell_2)\ell <_L r + (\ell_1\ell_2)\ell = r + (\ell_1\ell)\ell_2 <_L j + (i+1)\ell.$$

This is a contradiction because $(j + i\ell)_{i\in\mathbb{N}}$ is increasing whereas $\ell_1\ell_2 > 1$. Thus, any element $<_{\mathcal{L}}$-between $j + i\ell$ and $j + (i+1)\ell$ must be smaller than r; there are only finitely many such elements. ∎

To prove Theorem 4.3.5, we will extract the canonical word $\alpha_{\mathcal{L}} \in \{\omega, \omega^*, \mathbf{n}\}^\star$ of $\mathcal{L}$. For $t \leq j < t + \ell$, we can use Lemma 4.3.6 to decide in linear time whether the sequence $(j + i\ell)_{i\in\mathbb{N}}$ is in a copy of ω or ω^*.

By (4.1), we only need to determine the relative ordering of the sets $\{0, \ldots, t-1\}$ and $W_t, W_{t+1}, \ldots, W_{t+\ell-1}$. In the following we use W_s to denote the singleton $\{s\}$ for $s \in \{0, \ldots, t-1\}$. For $j, k \in \{0, , t+1, \ldots, t+\ell-1\}$, we fix the following terminologies:

- We say that W_j and W_k *interleave* if one of the following holds:
  - $j, k \in \{t, \ldots, t+\ell-1\}$ and W_j and W_k belong to the same copy of ω or ω^* in $\mathcal{L}$.
  - one of j, k belongs to $\{0, \ldots, t-1\}$ (say j) and the other (say k) belongs to $\{t, \ldots, t+\ell-1\}$, and j is $<_{\mathcal{L}}$ between two elements in W_k.
  - $j, k \in \{0, \ldots, t-1\}$ and there is $j' \in \{t, \ldots, t+\ell-1\}$ such that W_j and $W_{j'}$, W_k and $W_{j'}$ interleave.
- For $j, k \in \{0, \ldots, t+\ell-1\}$, we say that W_j is *to the left* of W_k if all elements in W_j are $<_{\mathcal{L}}$-below all elements in W_k.

Lemma 4.3.7 *For $j, k \in \{t, \ldots, t+\ell-1\}$, $j < k$, where W_j and W_k do not interleave. Then*

- *W_j is to the left of W_k if and only if $\Delta(q_j, (\diamond, 1)^{k-j}) \in F$.*
- *W_k is to the left of W_j if and only if $\Delta(q_j, (\diamond, 1)^{k-j}) \notin F$.*

Proof. Take $j, k \in \{t, \ldots, t+\ell-1\}$ where $j < k$ and say that W_j and W_k do not interleave. If W_j (W_k) is to the left of W_k (W_j), then it is immediate that $\Delta(q_j, (\diamond, 1)^{k-j})$ is (is not) an accepting state.

Conversely, suppose $\Delta(q_j, (\diamond, 1)^{k-j}) \in F$. Then $\forall i : j + i\ell <_{\mathcal{L}} k + i\ell$. We have the following cases:

(a) Assume both sequences $(j + i\ell)_{i\in\mathbb{N}}$ and $(k + i\ell)_{i\in\mathbb{N}}$ form a copy of ω^*. If $k + i_1\ell <_{\mathcal{L}} j + i_2\ell$ for some $i_1, i_2 \in \mathbb{N}$, then we have $k + i_1\ell <_{\mathcal{L}} j + i_2\ell <_{\mathcal{L}} k + i_2\ell$ and $i_1 > i_2$. But then we have

$$\cdots <_{\mathcal{L}} j + (2i_1 - i_2)\ell <_{\mathcal{L}} k + (2i_1 - i_2)\ell <_{\mathcal{L}} j + i_1\ell <_{\mathcal{L}} k + i_1\ell$$

and W_j and W_k interleave. Therefore for all i_1, i_2, $k + i_1\ell >_{\mathcal{L}} j + i_2\ell$ and W_j is to the left of W_k.

(b) Assume the sequence $(j + i\ell)_{i\in\mathbb{N}}$ forms a copy of ω^* and $(k + i\ell)_{i\in\mathbb{N}}$ forms a copy of ω. Then we have

$$\forall i, i' \in \mathbb{N} : j + i\ell \leq_{\mathcal{L}} j <_{\mathcal{L}} k \leq_{\mathcal{L}} k + i'\ell.$$

(c) Assume both sequences $(j+i\ell)_{i\in\mathbb{N}}$ and $(k+i\ell)_{i\in\mathbb{N}}$ form a copy of ω. If $k+i_1\ell <_{\mathcal{L}} j+i_2\ell$ for some $i_1, i_2 \in \mathbb{N}$, then we have $k + i_1\ell <_{\mathcal{L}} j + i_2\ell <_{\mathcal{L}} k + i_2\ell$ and $i_1 < i_2$. But then we will have

$$j + i_2\ell <_{\mathcal{L}} k + i_2\ell <_{\mathcal{L}} j + (2i_2 - i_1)\ell <_{\mathcal{L}} k + (2i_2 - i_1)\ell <_{\mathcal{L}} \cdots$$

and W_j and W_k interleave. Therefore for all i_1, i_2, $k + i_1\ell >_{\mathcal{L}} j + i_2\ell$ and W_j is to the left of W_k.

(d) Assume the sequence $(j + i\ell)_{i\in\mathbb{N}}$ forms a copy of ω and $(k + i\ell)_{i\in\mathbb{N}}$ forms a copy of ω^*. If $k + i_1\ell <_{\mathcal{L}} j + i_2\ell$ for some $i_1, i_2 \in \mathbb{N}$, then there is $i' \geq i_1$ such that

$$j + i'\ell <_{\mathcal{L}} k + i'\ell <_{\mathcal{L}} j.$$

This is in contradiction with the assumption on $(j + i\ell)_{i\in\mathbb{N}}$.

Now suppose $\Delta(q_j, (\diamond, 1)^{k-j}) \notin F$. Using a similar proof as above one can prove that W_k is to the left of W_j. ∎

Proof of Theorem 4.3.5. To extract the canonical word $\alpha_{\mathcal{L}}$, we first compute an equivalence relation $\sim$ on $\{0, \ldots, t + \ell - 1\}$ such that $j \sim k$ if W_j interleaves with W_k.

By Lemma 4.3.7, for $j, k \in \{t, t + 1, \ldots, t + \ell - 1\}$, W_j and W_k interleave if and only if there are $i_1, i_2 \in \mathbb{N}$ with $j + i_1\ell <_{\mathcal{L}} k + i_2\ell$ and $j_1, j_2 \in \mathbb{N}$ with $j + j_1\ell >_{\mathcal{L}} k + j_2\ell$. Hence, for $j \in \{0, \ldots, t + \ell - 1\}$ and $k \in \{t, \ldots, t + \ell - 1\}$, $j < k$, we have that $j \sim k$ if and only if

$$\exists c_1, c_2 \in \mathbb{N} : \Delta(q_j, (\diamond, 1)^{k-j+c_1\ell}) \in F \wedge \Delta(q_k, (\diamond, 1)^{j-k+c_2\ell}) \in F \tag{4.2}$$

Algorithm 7 computes the equivalence relation $\sim$. The correctness follows from (4.2).

Let $[j]_\sim$ be the $\sim$-equivalence class of j. For all $j \in \{0, \ldots, t + \ell - 1\}$, we compute a set $\mathrm{Left}([j]_\sim)$ defined as follows:

$$\mathrm{Left}([j]_\sim) = \{[k]_\sim \mid k \text{ is to the left of } j\}.$$

Lemma 4.3.7 implies an algorithm to compute $\mathrm{Left}([j]_\sim)$ for all $\sim$-equivalence classes; see Alg. 8

Algorithm 7 ClassifyLO($\mathcal{L}$)

for $j \in \{0, \ldots, t+\ell-1\}, k \in \{t, t+\ell-1\}, t < k$ **do**
  $q \leftarrow \Delta(q_j, (\diamond, 1)^{k-j})$.
  while q is not labeled "done for k" **do**
    if $q \in F$ **then** stop the while loop
    else label q "done for k" **end if**
    $q \leftarrow \Delta(q, (\diamond, 1)^{k-j})$
  end while
  if $q \notin F$ **then** stop this **for**-loop iteration.
  $q \leftarrow \Delta(q_k, (\diamond, 1)^{j-k+c\ell})$ where $c = \min\{c \mid j - k + c\ell > 0\}$.
  while q is not labeled "done for j" **do**
    if $q \in F$ **then** stop the while loop
    else label q "done for j" **end if**
    $q \leftarrow \Delta(q, (\diamond, 1)^{j-k})$
  end while
  if $q \in F$ **then** declare $j \sim k$
end for

Algorithm 8 ComputeLeft($\mathcal{L}$)

for $[j]_\sim, [k]_\sim$ where $j < k, [j]_\sim \neq [k]_\sim$ **do**
  if $\Delta(q_j, (\diamond, 1)^{k-j}) \in F$ **then**
    Put $[j]_\sim$ into Left($[k]_\sim$)
  else
    Put $[k]_\sim$ into Left($[j]_\sim$)
  end if
end for

We are now ready to give an algorithm for extracting $\alpha_{\mathcal{L}}$ from $\mathcal{A}$. We order all the $\sim$-equivalence classes $[j_1]_\sim, [j_2]_\sim, \ldots [j_r]_\sim$ in increasing order of the cardinalities of Left($[j_s]_\sim$). For each $[j_s]_\sim$, $1 \leq s \leq r$, decide the order type formed by the sets W_j where $j \in [j_s]_\sim$, using the condition given by Lemma 4.3.6. These order types can be considered as symbols taken from the alphabet $\{\omega, \omega^*, \mathbf{1}\}$. The concatenation of these symbols produces us a word $\alpha \in \{\omega, \omega^*, \mathbf{1}\}$ which corresponds to the order type of $\mathcal{L}$. However, α may not be the canonical word $\alpha_{\mathcal{L}}$. Therefore we "smooth" α: set α_L to be the result of replacing all 1ω in α to ω, all ω^*1 in α to ω^*, all sequences of $\mathbf{1}$s of length n to $\mathbf{n}$ and all $\omega^*\omega$ to ζ.

We now analyze the time complexity of the procedure that extracts $\alpha_{\mathcal{L}}$. Recall that the size of the input is measured as the number n of states in the automaton $\mathcal{A}$ that recognizes $\leq_{\mathcal{L}}$. In the ClassifyLO($\mathcal{L}$) algorithm, each state in $\mathcal{A}$ is visited at most n times (for each $(\diamond, 1)$-state out of q_j, it can be labeled "done for k" at most once for each $k \in \{t, \ldots, t + \ell - 1\}$). Hence, ClassifyLO($\mathcal{L}$) takes time $O(n^2)$. The ComputeLeft($\mathcal{L}$) algorithm visits each state in $\mathcal{A}$ at most once (for each j, k, it visits $\Delta(q_j, (\diamond, 1)^{k-j})$ at most once) and therefore takes $O(n)$ times. Sorting the equivalence classes $[j_1]_\sim, [j_2]_\sim, \ldots [j_r]_\sim$ and computing their order types also takes $O(n^2)$ time. Finally, computing the canonical word $\alpha_{\mathcal{L}}$ from α clearly takes $O(n^2)$ time. Hence, it takes $O(n^2)$ times to extract the canonical word $\alpha_{\mathcal{L}}$ from $\mathcal{A}$.

The algorithm for deciding the isomorphism problem takes two unary automatic linear orders $\mathcal{L}_1$ and $\mathcal{L}_2$ and compute their canonical words. By Lemma 4.3.3, $\mathcal{L}_1 \cong \mathcal{L}_2$ if and only if their canonical words coincides with each other. ∎

4.3.3 State complexity

Let $\mathcal{L} = (\mathbb{N}; \leq_{\mathcal{L}})$ be a unary automatic linear order. By Theorem 4.3.2, the order type of $\mathcal{L}$ is specified by the canonical word $\alpha_{\mathcal{L}} \in \{\zeta, \omega, \omega^*, \{\mathbf{n}\}_{n\in\mathbb{N}}\}^\star$. Let $m_{\mathcal{L}}$ be the number of instances of ω or ω^* in $\alpha_{\mathcal{L}}$ (where ζ is treated as $\omega^*\omega$) and let $k_{\mathcal{L}}$ be the sum of all n such that $\mathbf{n}$ appears in $w_{\mathcal{L}}$. We will express the state complexity of $\mathcal{L}$ in terms of the pair $(m_{\mathcal{L}}, k_{\mathcal{L}})$, whose size is defined to be $\max\{m_{\mathcal{L}}, k_{\mathcal{L}}\}$.

Theorem 4.3.8 *The (unary) state complexity of a unary automatic linear order $\mathcal{L} = (\mathbb{N}; \leq_{\mathcal{L}})$ is less than $2m_{\mathcal{L}}^2 + k_{\mathcal{L}}^2 + 2k_{\mathcal{L}}m_{\mathcal{L}} + k_{\mathcal{L}}$ and more than $2m_{\mathcal{L}}^2 - k_{\mathcal{L}}^2 + k_{\mathcal{L}}$.*

Proof. By Lemma 4.3.6, the optimal automaton $\mathcal{A}$ for $\mathcal{L}$ has $m_{\mathcal{L}} + k_{\mathcal{L}}$ $(1, 1)$-states: $k_{\mathcal{L}}$ many states on the $(1, 1)$-tail and $m_{\mathcal{L}}$ many states on the $(1, 1)$-loop. Each state on the $(1, 1)$-loop represents a copy of ω or ω^* in $\mathcal{L}$ and since this is the minimal automaton there is no interleaving. To specify whether each copy of ω or ω^* is increasing or decreasing and the relative ordering of copies of ω and $\omega^\star$, we need $2\ell = 2m_{\mathcal{L}}$ $(1, \diamond)$ or $(\diamond, 1)$ states off each $(1, 1)$-loop state. To specify the ordering of the singleton elements represented by the states on the $(1, 1)$-tail with respect to each other and to copies of ω, ω^*, we need up to

$2(k_{\mathcal{L}} - j + m_{\mathcal{L}} - 1)$ states off q_j, $j \in \{0, ..., k_{\mathcal{L}} - 1\}$. Therefore, an upper bound to the number of states in an optimal unary automaton is

$$m_{\mathcal{L}} + k_{\mathcal{L}} + m_{\mathcal{L}}(2m_{\mathcal{L}}) + \sum_{j=0}^{k_{\mathcal{L}}-1} 2\,(k_{\mathcal{L}} - j + m_{\mathcal{L}} - 1) = 2m_{\mathcal{L}}^2 + k_{\mathcal{L}}^2 + 2k_{\mathcal{L}}m_{\mathcal{L}} + m_{\mathcal{L}}.$$

For a pair $(m_{\mathcal{L}}, k_{\mathcal{L}})$, we can easily find a linear order with parameters $(m_{\mathcal{L}}, k_{\mathcal{L}})$ whose state complexity is $2m_{\mathcal{L}}^2 + k_{\mathcal{L}}^2 + 2k_{\mathcal{L}}m_{\mathcal{L}} + m_{\mathcal{L}}$. For example, when $m_{\mathcal{L}} > 0$, the linear order $\omega^*\mathbf{k}_{\mathcal{L}}\omega^{m_{\mathcal{L}}-1}$ matches this upper bound. ∎

Example 4.3.9 *A optimum automaton representing the order relation of the linear order $\omega + 1 + \omega^* + \omega^*$ (with canonical word $\omega 1 \omega^* \omega^*$ is presented in Figure 4.4. Note that the automaton has 20 states, which is between the lowerbound (17) and upperbound (26) as stated in Theorem 4.3.8*

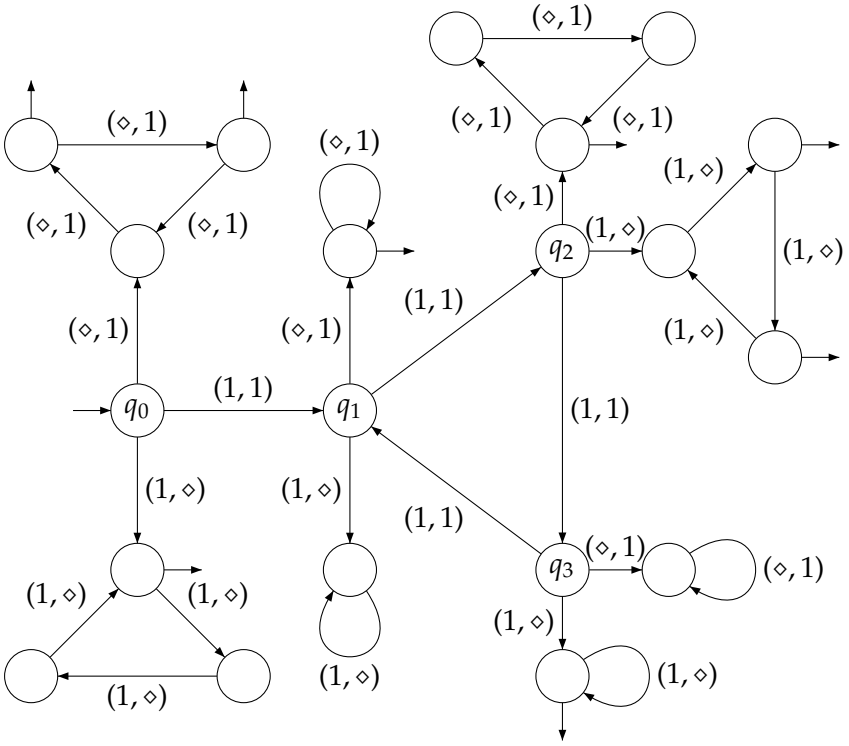


Figure 4.4: An optimal automaton for $\omega + 1 + \omega^* + \omega^*$.

Corollary 4.3.10 *The (unary) state complexity for the class of unary automatic linear orders is quadratic in the size of the associated parameter.*

4.4 Unary automatic equivalence structures

4.4.1 A characterization theorem

We use $\mathcal{E} = (\mathbb{N}; E)$ to denote an equivalence structure where $E \subseteq \mathbb{N}^2$ is an equivalence relation. The following can be proved in a similar way as Proposition 4.3.1.

Proposition 4.4.1 *The membership problem for automatic equivalence structures is decidable in time $O(n^3)$.*

Blumensath[6] and Khoussainov/Rubin[74] described the structure of unary automatic equivalence structures.

Theorem 4.4.2 ([6, 74]) *An equivalence structure has a unary automatic presentation if and only if it has finitely many infinite equivalence classes and there is a finite bound on the sizes of the finite equivalence classes.*

The *height* of an equivalence structure $\mathcal{E}$ is a function $h_{\mathcal{E}} : \mathbb{N} \cup \{\infty\} \to \mathbb{N} \cup \{\infty\}$ such that $h_{\mathcal{E}}(x)$ is the number of E-equivalence classes of size x. Two equivalence structures $\mathcal{E}_1$ and $\mathcal{E}_2$ are isomorphic if and only if $h_{\mathcal{E}_1} = h_{\mathcal{E}_2}$. By Theorem 4.4.2, the function $h_{\mathcal{E}}$ for unary automatic equivalence structure $\mathcal{E}$ is finitely nonzero.

Corollary 4.4.3 *The isomorphism problem for unary automatic equivalence structures is decidable.*

Proof. For each $n \in \mathbb{N}$, define the formula $\text{size}_n(x)$ as

$$\exists y_1 \cdots \exists y_{n-1} : \bigwedge_{i=1}^{n-1} (y_i \neq x \wedge E(y_i, x)) \wedge \forall z : (\bigwedge_{i=1}^{n-1} z \neq y_i \wedge z \neq x) \to \neg E(x, z).$$

Let $H = h_{\mathcal{E}}(\infty)$ and $H_n = h_{\mathcal{E}}(n)$. By Theorem 4.4.2, any unary automatic equivalence structure $\mathcal{E}$ with height $h_{\mathcal{E}}$ can be defined by the sentence $\varphi_{\mathcal{E}}$ that is the conjunction of the following:

$$\exists x_1 \cdots \exists x_H : \left[\bigwedge_{i=1}^{H} \exists^\infty y : E(x_i, y) \ \wedge \bigwedge_{i,j=1; i \neq j}^{H} \neg E(x_i, x_j) \right.$$
$$\left. \wedge \ \forall x \exists^\infty y : E(x, y) \to \bigvee_{i=1}^{H} E(x, x_i) \right] \ \wedge \bigwedge_{n : H_n = \infty} \left[\exists^\infty x : \text{size}_n(x) \right]$$

and

$$\bigwedge_{m,n : H_n = m} \exists y_1 \cdots \exists y_m : \bigwedge_{i=1}^{m} \text{size}_n(y_i) \wedge \bigwedge_{i,j=1; i \neq j}^{m} \neg E(y_i, y_j)$$

For any equivalence structure $\mathcal{E}_1$, $\mathcal{E}_1 \cong \mathcal{E}$ if and only if $\mathcal{E}_1 \models \varphi_{\mathcal{E}}$. By Theorem 2.5.11, the isomorphism problem is decidable. ∎

The sentence $\varphi_{\mathcal{E}}$ contains two alternations of quantifiers[1], each causes an exponential blow-up in the size of the automaton corresponding to $\varphi_{\mathcal{E}}$ [70]. This implies that the decision procedure given by Corollary 4.4.3 has doubly exponential runtime in the sizes of the input automata. As in the case of linear orders, the decision procedure requires $\varphi_{\mathcal{E}}$ to be known beforehand and is thus non-uniform in the input automata. We now present a uniform solution to the isomorphism problem that significantly improves the time complexity.

4.4.2 An efficient solution to the isomorphism problem

Theorem 4.4.4 *The isomorphism problem for unary automatic equivalence structures is decidable in linear time in the sizes of the input automata.*

Let $\mathcal{E}$ be recognized by a unary automaton $\mathcal{A} = (S, \Delta, I, F)$ with n states. Recall the definitions of t, ℓ, and q_j from Section 4.1.2. Observe that each $j < t + \ell$ belongs to an infinite equivalence class if and only if there is an accepting state on the $(\diamond, 1)$ loop from q_j. Choose $j \in \{t, \ldots, t + \ell - 1\}$. If j belongs to an infinite equivalence class then for all $i \in \mathbb{N}$, $j + i\ell$ is in an infinite equivalence class. By Theorem 4.4.2, there are only finitely many infinite equivalence classes in $\mathcal{E}$. Hence, for some i and k, $i \neq k$, $(j+i\ell, j+k\ell) \in E$, i.e., $\Delta(q_j, (\diamond, 1)^{(k-i)\ell})$ is an accepting state.

Let $\gamma_j > 0$ be the least number such that $\Delta(q_j, (\diamond, 1)^{\gamma_j \ell}) \in F$. Recall the definition of W_j from Section 4.3.2.

Lemma 4.4.5 *The set W_j is partitioned into γ_j infinite equivalence classes.*

Proof. By minimality of γ_j, elements in $\{j, j + \ell, \ldots, j + (\gamma_j - 1)\ell\}$ are pairwise non E-equivalent. Moreover for each i, $(j + i\ell, j + (i + \gamma_j)\ell) \in E$. ∎

Lemma 4.4.6 *If j belongs to an infinite equivalence class C, then for all $k \in \{t, \ldots, t + \ell - 1\}$ where $W_k \cap C \neq \emptyset$, $\gamma_k = \gamma_j$.*

Proof. Observe that for $k \in \{t, \ldots, t + \ell - 1\}$, $W_k \cap C \neq \emptyset$ if and only if $\Delta(q_j, (\diamond, 1)^c) \in F$ where $c = k - j + d\ell$ for some $d \in \mathbb{N}$. This means that for all $i \in \mathbb{N}$, $(j + i\ell, k + (d + i)\ell) \in E$. Since for $i, i' \in \{0, \ldots, \gamma_j - 1\}$, $i \neq i'$, $(j + i\ell, j + i'\ell) \notin E$, we have $(k + (d + i)\ell, k + (d + i')\ell) \notin E$ and hence

$$\forall i \in \{0, \ldots, \gamma_j - 1\} : \Delta(q_k, (\diamond, 1)^{i\ell}) \notin F.$$

Also, since $(k + d\ell, k + (d + \gamma_j)\ell) \in E$, we have $\Delta(q_k, (\diamond, 1)^{\gamma_j}) \in F$. By definition, this means $\gamma_k = \gamma_j$. ∎

[1] The counting quantifier $\exists^{\infty} x : \psi(x)$ is treated as $\forall x \exists y >_{\text{llex}} x : \psi(x)$

By Lemma 4.4.6, we define an equivalence relation $\sim$ on $\{t, \ldots, t+\ell-1\}$ such that

- $j \sim k$ implies that the sets W_j and W_k are both partitioned into the same γ_j infinite equivalence classes;
- $j \nsim k$ implies that the equivalence classes containing elements in W_j are disjoint from the ones containing elements in W_k.

Using this fact, we define an algorithm that computes the value of $h_{\mathcal{E}}(\infty)$: the number of infinite equivalence classes in $\mathcal{E}$. We first compute the equivalence relation $\sim$ by reading the final states on the $(\diamond, 1)$-tail and -loop out of each q_j, $t \leq j < t+\ell$. Then for each $\sim$-equivalence class $[j]_\sim$, compute the value of γ_j and add it to $h_{\mathcal{E}}(\infty)$. To compute γ_j, we examine $\Delta(q_j, (\diamond, 1)^{d\ell})$ for increasing values of d until we find an accepting state or repeat a state. See Algorithm 9.

The algorithm examines each state in $\mathcal{A}$ at most twice (once for computing $\sim$ and the second time for computing γ_j). Hence Algorithm 9 takes time $O(n)$.

Algorithm 9 InfClass($\mathcal{E}$)

for $j \in \{t, \ldots, t+\ell-1\}$ where the $(\diamond, 1)$-loop off q_j contains a final state **do**
  for all final states q on the $(\diamond, 1)$-tail and -loop off q_j **do**
    Let $k \in \{t, \ldots, t+\ell-1\}$ be such that $q = \Delta(q_j, (\diamond, 1)^c)$ where $c \equiv k - j \mod \ell$.
    Declare $j \sim k$.
  end for
end for
$h \leftarrow 0$
for all equivalence classes $[j]_\sim$ **do**
  $c \leftarrow 1$; $q \leftarrow \Delta(q_j, (\diamond, 1)^{c\ell})$
  while $q \notin F$ and q is not marked "processed" **do**
    Mark q as "processed"
    $c \leftarrow c + 1$; $q \leftarrow \Delta(q_j, (\diamond, 1)^{c\ell})$
  end while
  if $q \in F$ **then** $h \leftarrow h + c$ **end if**
end for
return h.

We now consider the finite equivalence classes. Take $j \in \{t, \ldots, t+\ell-1\}$ such that the state q_j has no accepting state on its $(\diamond, 1)$-loop. Note that by Lemma 4.4.5, $\Delta(q_j, (\diamond, 1)^{c\ell}) \notin F$ for any $c \in \mathbb{N}$. Hence,

$$\forall i, i' \in \mathbb{N} : i \neq i' \Rightarrow (j + i\ell, j + i'\ell) \notin E. \tag{4.3}$$

By (4.3), for any $k \in \{t, \ldots, t+\ell-1\}$, $j \neq k$, there is at most one final state q on the $(\diamond, 1)$-tail out of q_j such that $q = \Delta(q_j, (\diamond, 1)^c)$ for some $c \equiv k - j \mod \ell$.

Definition 4.4.7 *A* corresponding set *is a tuple* $(j_1, \ldots, j_m)$ *where each* q_{j_i}, $1 \le i \le m$, *is a* $(1,1)$*-loop and for each* q_{j_i} *and* $s \in \{1, \ldots, m-i\}$ *there is* $c \equiv (j_{s+i} - j_i) \mod \ell$ *such that the state*

$$r_s^i = \Delta(q_{j_i}, (\diamond, 1)^c) \in F;$$

moreover, these are on the only accepting states on the $(\diamond, 1)$*-tail off* q_{j_i}. *A corresponding set is* maximal *if it is not a subset of a larger corresponding set.*

In the following, for $t \le j < t + \ell$, we let ℓ_j and t_j be the lengths of the $(\diamond, 1)$-loop and -tail off q_j, respectively. Let $p = \max\{t_j + \ell_j \mid t \le j < t + \ell\}$.

Lemma 4.4.8 *For any* k, $h_{\mathcal{E}}(k) = \infty$ *if and only if there is a maximal corresponding set of size* k.

Proof. Suppose $j_1, \ldots, j_k$ form a maximal corresponding set. Let m_{s+i}^i be such that

$$r_s^i = \Delta(q_{j_i}, (\diamond, 1)^{(j_{s+i} - j_i) + m_{s+i}^i \ell}) \in F.$$

Then for each $c \ge p$, $\{j_1 + c\ell, j_2 + (c + m_2^1)\ell, \ldots, j_k + (c + m_k^1)\ell\}$ is an equivalence class of size k. Note that here we require $c \ge p$ since for small values of c, the equivalence class of $j_1 + c\ell$ may also contain elements from $\{0, \ldots, t-1\}$.

On the other hand, suppose there are infinitely many $\mathcal{E}$-equivalence classes of size k. Consider an equivalence class $\{x_1, \ldots, x_k\}$ where $p \le x_1 < x_2 < \cdots < x_k$. For $1 \le i < k$, define $j_i \in \{t, \ldots, t + \ell - 1\}$ be such that $x_i \equiv j_i \mod \ell$. By (4.3), $j_1, \ldots, j_k$ is a maximal corresponding set. ∎

Lemma 4.4.8 implies an algorithm for computing the set $\{k \in \mathbb{N}_+ \mid h_{\mathcal{E}}(k) = \infty\}$. See Algorithm 10. It is clear that a state is visited at most once in Algorithm 10 and hence the algorithm takes time $O(n)$.

Algorithm 10 FinClass($\mathcal{E}$)

```
for j ∈ {t, ..., t + ℓ − 1} do
    for all accepting states q on the (⋄, 1)-tail off q_j do
        Let k ∈ {t, ..., t + ℓ − 1} be such that q = Δ(q_j, (⋄, 1)^c) where c ≡ k − j (mod ℓ).
        Declare that j and k are in the same corresponding set.
    end for
end for
for all corresponding sets C do
    Declare h_E(|C|) = ∞
end for
```

Proof of Theorem 4.4.4. To decide whether two unary automatic equivalence structures $\mathcal{E}_1, \mathcal{E}_2$ are isomorphic we first use the unary automata recognizing E_1 and E_2 to compute

their height functions and then check if $h_{\mathcal{E}_1} = h_{\mathcal{E}_2}$. Hence, we begin by giving an algorithm for extracting the height function of a unary automatic equivalence structure $\mathcal{E}$ from a unary automaton $\mathcal{A}$.

The procedure first runs Algorithm 9 to compute the value $h_{\mathcal{E}}(\infty)$, and runs Algorithm 10 to compute all k with $h_{\mathcal{E}}(k) = \infty$. Both algorithms take $O(n)$ time.

It only remains to compute the sizes of equivalence classes for elements in $\{0, \ldots, t-1\}$, which requires reading through the $(\diamond, 1)$-tails off the $(1, 1)$-tail. Again this step has runtime $O(n)$.

In summary, the algorithm that computes $h_{\mathcal{E}}$ from $\mathcal{A}$ has runtime $O(n)$. Note that the domain of $h_{\mathcal{E}}$ is a subset of $\{1, \ldots, n, \infty\}$ so comparing it with $h_{\mathcal{E}'}$ takes linear time. Therefore, the isomorphism problem for unary automatic equivalence relations is solved in linear time in the maximum of the sizes of the input automata. ∎

4.4.3 State complexity

Given a unary automatic equivalence structure $\mathcal{E} = (\mathbb{N}; E)$, we want to define the optimal unary automaton for $\mathcal{E}$. We will express the state complexity in terms of the height function $h_{\mathcal{E}}$; define the size of $h_{\mathcal{E}}$ to be

$$|h_{\mathcal{E}}| = \sum_{n : h_{\mathcal{E}}(n) < \infty} n h_{\mathcal{E}} + n_{\text{inf}} + h_{\text{inf}}$$

where $h_{\text{inf}} = h_{\mathcal{E}}(\infty)$ and $n_{\text{inf}} = \sum_{n : h_{\mathcal{E}}(n) = \infty} n$.

Lemma 4.4.9 *Let $\mathcal{A}$ be a unary automaton recognizing $\mathcal{E}$, then $n_{\text{inf}} \leq \ell$.*

Proof. For any n, $h_{\mathcal{E}}(n) = \infty$ if and only if there are $t \leq j_1 < j_2 < \cdots < j_n < t + \ell$ such that $\Delta(q_{j_1}, (\diamond, 1)^{j_i - j_1}) \in F$ for all $i = 1, \ldots, n$ and no other $(\diamond, 1)$ states off q_i are accepting. These q_{j_i} may not be shared among disjoint equivalence classes, hence $n_{\text{inf}} \leq \ell$. ∎

Theorem 4.4.10 *The state complexity of any unary automatic equivalence structures $\mathcal{E} = (\mathbb{N}; E)$ is at least*

$$\sum_{n : h_{\mathcal{E}}(n) < \infty} n^2 h_{\mathcal{E}}(n) + 2h_{\text{inf}}(n_{\text{inf}} + 1) + n_{\text{inf}} + 1$$

and at most

$$\sum_{n : h_{\mathcal{E}}(n) < \infty} n^2 h_{\mathcal{E}}(n) + \sum_{n : h_{\mathcal{E}}(n) = \infty} n^2 + 2h_{\text{inf}}(n_{\text{inf}} + 1) + 1.$$

Proof. We say a collection of $(1, 1)$-states $\{r_1, \ldots, r_m\}$ in $\mathcal{A}$ *represents* an E-equivalence class K if for each $x \in K$, $\Delta(q_{\text{init}}, (1, 1)^x) = r_i$ for some $1 \leq i \leq m$, where q_{init} is the initial state. Let K be a finite equivalence class. It must be represented by some $\{r_1, \ldots, r_m\}$ where there are $m - i$

accepting states on the $(\diamond, 1)$-tail off r_i. In an optimal unary automaton recognizing $\mathcal{E}$, the length of the $(\diamond, 1)$-tails off r_i states is minimized by arranging the $r_1, \ldots, r_j$ consecutively. In this case, the tail off r_i contains $m - i$ states; by symmetry, the number of $(\diamond, 1)$ and $(1, \diamond)$ states associated to the class K is $2\sum_{i=1}^{|K|}(|K| - i) = |K|^2 - |K|$. Counting the $(1, 1)$ states representing K, there are $|K|^2$ states associated to K. Note that in the optimal automaton, if there are infinitely many equivalence classes of the same size, they are all represented by the same $(1, 1)$ states.

If each infinite equivalence class of $\mathcal{E}$ is represented by a single state on the $(1, 1)$-loop of an automaton $\mathcal{A}$, then $\ell > h_{\text{inf}}$. Moreover, the $(\diamond, 1)$-loop out of with each such state must have size at least ℓ. In this case, $\ell + 1$ states are associated with each infinite equivalence class. One may hope to reduce the number of states by using multiple states $r_1, \ldots, r_k$ to represent an infinite equivalence class K. In this case, k must be a divisor of ℓ and the $(\diamond, 1)$-loop out of each r_i has length ℓ/k. Thus, at least $k + k(\ell/k) = k + \ell$ states are associated with K, which is no improvement. However, we can reduce the number of states by using a single $(1, 1)$-loop state r to represent all the (finitely many) infinite components. To do so, define a $(\diamond, 1)$-loop (respectively, $(1, \diamond)$-loop) out of r with length $h_{\text{inf}}\ell$ and a single accepting state, $\Delta(r, (\diamond, 1)^{h_{\text{inf}}\ell})$. With this representation, $1 + 2h_{\text{inf}}\ell$ states are used for all the infinite equivalence classes (as opposed to more than $h_{\text{inf}}^2 + h_{\text{inf}}$). By Lemma 4.4.9 and the above discussion, the smallest possible length for the $(1, 1)$-loop is $(n_{\text{inf}} + 1)$. For each n such that $h_{\mathcal{E}}(n) = \infty$, there are $n^2 - n$ $(1, \diamond)$- and $(\diamond, 1)$-states off the $(1, 1)$-loop. Thus, there must be at least

$$1 + 2h_{\text{inf}}\ell + \sum_{n:h_{\mathcal{E}}(n)=\infty} n^2$$

states on the $(1, 1)$-loop and its peripheries.

We can define an automaton $\mathcal{A}$ that recognizes $\mathcal{E}$. The $(1, 1)$-tail of $\mathcal{A}$ has length $\sum_{n:h_{\mathcal{E}}(n)<\infty} nh_{\mathcal{E}}(n)$ and its $(1, 1)$-loop has length $n_{\text{inf}} + 1$. The total size of $\mathcal{A}$ is

$$\sum_{n:h_{\mathcal{E}}(n)<\infty} n^2 h_{\mathcal{E}}(n) + 1 + 2h_{\text{inf}}(n_{\text{inf}} + 1) + \sum_{n:h_{\mathcal{E}}(n)=\infty} n^2.$$

An optimal automaton must have at most this many states.

To obtain a lower bound, we note that there may be overlap between states in the $(1, 1)$-loop representing equivalence classes of different sizes, some of which occur infinitely often and others which do not. In particular, this can only occur for E-equivalence classes K, K' where $|K| > |K'|$ and $h_{\mathcal{E}}(|K|) = \infty$, $h_{\mathcal{E}}(|K'|) < \infty$. With optimal overlapping, the minimum number of states in a unary automaton recognizing $\mathcal{E}$ is

$$\sum_{n:h_{\mathcal{E}}(n)<\infty} n^2 h_{\mathcal{E}}(n) + \sum_{n:h_{\mathcal{E}}(n)=\infty} n^2 + 2h_{\text{inf}}(n_{\text{inf}} + 1) + 1 - c$$

for some c. Moreover, c is not more than the number of all $(\diamond, 1)$- and $(1, \diamond)$- states associated with finite equivalence classes occurring infinitely often, so $c < \sum_{n:h_{\mathcal{E}}(n)=\infty}(n^2 - n)$, as required. ∎

Corollary 4.4.11 *The (unary) state complexity for the class of unary automatic equivalence structure is quadratic in terms of the height function.*

4.5 Unary automatic trees

4.5.1 Characterizing unary automatic trees

This section studies unary automatic trees $\mathcal{T} = (\mathbb{N}; \leq_{\mathcal{T}})$. An *anti-chain* in $\mathcal{T}$ is a set of nodes that are pairwise incomparable. It is clear that an $O(n^3)$ algorithm checks whether a given automaton recognizes a partial order. Checking if $\leq_{\mathcal{T}}$ is total on every set of predecessors $(\forall x, y, z : y, z \leq_{\mathcal{T}} x \rightarrow y \leq_{\mathcal{T}} z \vee z \leq_{\mathcal{T}} y)$ takes time $O(n^4)$. Checking the existence of a root (least element) may take exponential-time because of the impact of alternation of quantifiers on the size of the automaton for the query. We improve this exponential bound when $\leq_{\mathcal{T}}$ is recognized by a unary (rather than arbitrary) automaton.

Lemma 4.5.1 *There is an $O(n)$ time algorithm that checks if a unary automatic partial order $(\mathbb{N}; \leq_{\mathcal{T}})$ has a least element.*

Proof. Suppose $\leq_{\mathcal{T}}$ is recognized by unary automaton $\mathcal{A} = (S, \Delta, \{q_{\text{init}}\}, F)$ with parameters t, ℓ (as in Section 4.1.2). If there is a least element x then $x < t + \ell$. Indeed, if $x \geq t + \ell$, there is $t \leq y < t + \ell$ such that $q = \Delta(q_{\text{init}}, (1, 1)^x) = \Delta(q_{\text{init}}, (1, 1)^y)$. By definition of x, $x <_{\mathcal{T}} y$ and $x <_{\mathcal{T}} 2x - y$. Therefore, $\Delta(q, (\diamond, 1)^{y-x}) \in F$ and $\Delta(q, (\diamond, 1)^{x-y}) \in F$. But, this implies that $y <_{\mathcal{T}} x$, a contradiction. Thus, to check for a root it is sufficient to check if each of $\{0, \ldots, t + \ell - 1\}$ is the root and this procedure examines each state of $\mathcal{A}$ at most once. In particular, Algorithm 11 does this and runs in $O(n)$. ∎

The following proposition follows immediately from Lemma 4.5.1.

Proposition 4.5.2 *The membership problem for unary automatic trees is decidable in time $O(n^4)$.*

As we saw in previous sections, a good characterization of a class of unary automatic structures may lead to a better understanding of complexity bounds. We present such a characterization of unary automatic trees (in the spirit of Theorem 4.1.3). A *parameter set* Γ is a tuple $(\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ where $\mathcal{T}_0, \ldots, \mathcal{T}_m$ are finite trees (with disjoint domains T_i), $\sigma : \{1, \ldots, m\} \rightarrow T_0$ and $X : \{1, \ldots, m\} \rightarrow \{\emptyset\} \cup \bigcup_i T_i$ such that $X(i) \in T_i \cup \{\emptyset\}$.

Definition 4.5.3 *A* tree-unfolding *of a parameter set Γ is the tree* $\text{UF}(\Gamma)$ *defined as follows:*

Algorithm 11 MinElement($\mathcal{A}$)

```
Initialize the list L = 0, ..., t + ℓ − 1.
while L ≠ ∅ do
    Let j be the first element in L.
    if all (⋄, 1)-states out of q_j are accepting then
        q_j is the R-least element; return true
    else delete j from L. end if
    for k ∈ L do
        if Δ(q_j, (1, ⋄)^{k−j}) is accepting then delete k from L. end if
    end for
end while
return false
```

- UF(Γ) *contains one copy of* $\mathcal{T}_0$ *and infinitely many copies of each* $\mathcal{T}_i$ $(1 \le i \le m)$, $(\mathcal{T}_i^j)_{j\in\omega}$. *If* $x \in T_i$, *its copy in* $\mathcal{T}_i^j$ *is denoted by* (x, j)

- *For* $i \in \{1, \ldots, m\}$, *if* $X(i) \neq \emptyset$, *the root of* $\mathcal{T}_i^0$ *is a child of* $\sigma(i)$, *and the root of* $\mathcal{T}_i^{j+1}$ *is a child of* $(X(i), j)$ *for all* j.

- *For* $i \in \{1, \ldots, m\}$, *if* $X(i) = \emptyset$, *the root of* $\mathcal{T}_i^j$ *is a child of* $\sigma(i)$ *for all* j.

Theorem 4.5.4 *A tree* $\mathcal{T}$ *is unary automatic if and only if* $\mathcal{T} \cong \mathrm{UF}(\Gamma)$ *for some parameter set* $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$.

Suppose $\mathcal{T} = (\mathbb{N}; \le_{\mathcal{T}})$ is recognized by a unary automaton $\mathcal{A}$ with n states and parameters t, ℓ. Recall the definition of W_j, $0 \le j < t + \ell$, from Section 4.3.2. We say that two disjoint sets X and Y of nodes in $\mathcal{T}$ are *incomparable* if $\forall x \in X \forall y \in Y : x|_{\mathcal{T}} y$.

Lemma 4.5.5 *For* $j \in \{t, t + \ell - 1\}$, *the set* W_j *forms either an anti-chain or finitely many pairwise incomparable infinite chains in* $\mathcal{T}$.

Proof. If there is no c such that $\Delta(q_j, (\diamond, 1)^{c\ell})$ is accepting, then the sequence $(j + i\ell)_{i\in\mathbb{N}}$ is an anti-chain. Otherwise, let n_j be the least such c. In this case, W_j is partitioned into exactly n_j pairwise incomparable chains in $\mathcal{T}$. Indeed, $j + m\ell <_{\mathcal{T}} j + (m + in_j)\ell$ for all i and for $0 \le m < n_j$, thus making $(j + (m + in_j)\ell)_{i\in\mathbb{N}}$ an infinite chain; furthermore elements in $\{j, j + \ell, \ldots, j + (n_j - 1)\ell\}$ are pairwise incomparable. ∎

By Lemma 4.5.5, let

$$A = \{j \mid W_j \text{ forms an anti-chain}, t \le j < t + \ell\}$$
$$C = \{t, \ldots, t + \ell - 1\} - A$$

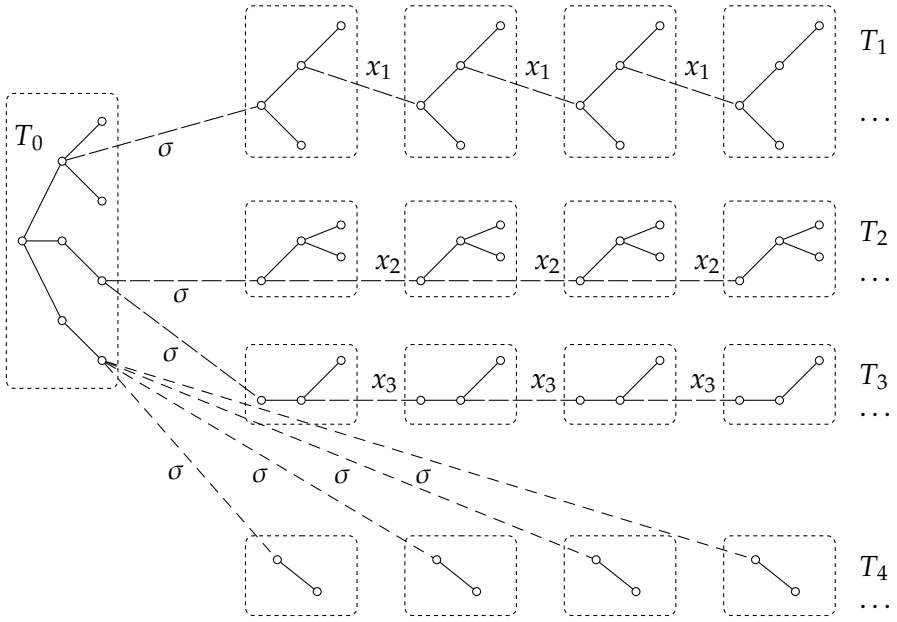


Figure 4.5: An example of a tree-unfolding.

For each $j \in C$, let n_j be the number of infinite chains in W_j. For $0 \leq m < n_j$, we denote the infinite chain formed by $(j + (m + in_j)\ell)_{i\in\mathbb{N}}$ by $W_{j,m}$.

We consider the circumstances in which two chains $W_{j,s}$ and $W_{k,s'}$ belong to the same infinite path in $\mathcal{T}$ (they *interleave* in the sense of Section 4.3). Fix $j, k \in C$ where $j \neq k$. If there is no m such that $\Delta(q_j, (\diamond, 1)^{k-j+m\ell})$ is accepting, then no $W_{j,s}$ and $W_{k,s'}$ interleave. Otherwise, let m be the least number such that $\Delta(q_j, (\diamond, 1)^{k-j+m\ell}) \in F$.

Lemma 4.5.6 *With m defined as above, we have $n_j = n_k$ and W_j and W_k form exactly n_j pairwise incomparable infinite chains.*

Proof. By assumption, $\Delta(q_j, (\diamond, 1)^{k-j+m\ell})$ is accepting. Hence, $j <_{\mathcal{T}} k+m\ell$ and $k+m\ell \in W_{k,m_1}$ for some $m_1 \in \{0, \ldots, n_k - 1\}$. Therefore, $m_1 \equiv m \mod n_k$ and $W_{j,0}, W_{k,m_1}$ interleave in $\mathcal{T}$. Similarly, since $j + n_j\ell <_{\mathcal{T}} k + n_j\ell + m\ell$, there is $0 \leq m_2 < n_k$ such that $m_2 \equiv n_j + m \mod n_k$ and $W_{j,0}, W_{k,m_2}$ interleave in $\mathcal{T}$. Therefore, W_{k,m_1}, W_{k,m_2} interleave and by definition this implies $m_1 = m_2$. Hence, $n_j = cn_k$ for some $c > 0$. Since there is some interleaving between j and k sets, there is r such that $\Delta(q_k, (\diamond, 1)^{j-k+r\ell}) \in F$. Repeating the above argument with the roles of j and k reversed, we see that $n_k = c'n_j$ for some $c > 0$. Thus, $n_j = n_k$ and the union of $(j + i\ell)_{i\in\mathbb{N}}$ and $(k + i\ell)_{i\in\mathbb{N}}$ contains exactly n_j pairwise disjoint infinite chains: for all $i \in \{0, \ldots, n_j - 1\}$, $W_{j,i}$ and $W_{k,m'}$ interleave if and only if $m' = m + i \mod n_j$. ∎

Any infinite path through $\mathcal{T}$ must be given by element(s) in C. Therefore, Lemma 4.5.6 implies that $\mathcal{T}$ contains only finitely many infinite paths. We define a *component* of $\mathcal{T}$ as a connected subgraph of $\mathcal{T}$ which contains exactly one infinite path and such that all the elements in the subgraph are greater than or equal to t. Fix $j \in C$ and $k \in A$. By Lemma 4.5.6, $(j+i\ell)_{i\in\mathbb{N}}$ belongs to exactly n_j components, $B_0, \ldots, B_{n_j-1}$. If there is no m such that $\Delta(q_j, (\diamond, 1)^{k-j+m\ell})$ is accepting, then no element in the anti-chain $(k + i\ell)_{i\in\mathbb{N}}$ belongs to any B_r. Otherwise, let m be the least such that $\Delta(q_j, (\diamond, 1)^{k-j+m\ell}) \in F$. Each $j+i\ell$ has $k+(i+m)\ell$ as a descendent. Therefore $(k + i\ell)_{i\in\mathbb{N}}$ is partitioned into a finite set $\{k + i\ell \mid 0 < i < m\}$ and exactly n_j infinite classes $\{(k + (m + s + in_j)\ell)_{i\in\mathbb{N}} \mid s = 0, \ldots, n_j - 1\}$, each belonging to a unique B_r.

We have considered the case when at least one of j and k is in C. Now, suppose $j, k \in A$ and neither $(j + i\ell)_{i\in\mathbb{N}}$ nor $(k + i\ell)_{i\in\mathbb{N}}$ intersects with any component of $\mathcal{T}$. If there is m such that $\Delta(q_j, (\diamond, 1)^{k-j+m\ell}) \in F$, then the union $(j+i\ell)_{i\in\mathbb{N}} \cup (k+i\ell)_{i\in\mathbb{N}}$ is a subset of infinitely many disjoint finite subtrees in $\mathcal{T}$, each of which contains the nodes $j+i\ell$ and $k+(i+m)\ell$ for some i. We call these disjoint finite trees *independent*.

The above argument facilitates the definition of an equivalence relation $\sim$ on $\{t, \ldots, t + \ell - 1\}$ as $j \sim k$ if and only if

1. $j \in C$ (or $k \in C$) and $\{j + i\ell\}_{i\in\mathbb{N}}$ and $\{k+i\ell\}_{i\in\mathbb{N}}$ belong to the same n_j (or n_k) components in $\mathcal{T}$; or,
2. $j, k \in A$ and there is $h \in C$ such that $j \sim h$ and $k \sim h$;
3. $j, k \in A$ and $\{j + i\ell\}_{i\in\mathbb{N}}$ and $\{k + i\ell\}_{i\in\mathbb{N}}$ belong to the same collection of independent trees in $\mathcal{T}$.

We use $[j]_\sim$ to denote the $\sim$-equivalence class of j.

Theorem 4.5.4 We now show that any unary automatic tree is isomorphic to the tree-unfolding UF(Γ) of some parameter set $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$. For each $\sim$-equivalence class $[j]_\sim$, either $[j]_\sim$ represents infinitely many independent trees or $[j]_\sim$ represents finitely many components of $\mathcal{T}$.

In the first case, the independent trees represented by $[j]_\sim$ are pairwise isomorphic. Moreover, the set of ancestors of these independent trees in $\mathcal{T}$ is finite because they are not in a component of $\mathcal{T}$. In the second case, the components of $\mathcal{T}$ represented by $[j]_\sim$ are pairwise isomorphic. Each of these components can be described by "unfolding" a finite graph, $\mathcal{T}_c$, of size $|[j]_\sim|$: each $k \sim j$ contributes one vertex to $\mathcal{T}_c$ and the edges are specified by the relations between $(j + i\ell)_{i\in\mathbb{N}}$, $(k + i\ell)_{i\in\mathbb{N}}$ discussed above; the root of a later copy of $\mathcal{T}_c$ is a child of a fixed node in the immediately preceding copy. Observe that, as in the first case, the set of ancestors in $\mathcal{T}$ of the component is finite. It is immediate to translate this description to an appropriate parameter set for Γ.

Conversely, we will show that if $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ is a parameter set, UF(Γ) is a unary automatic tree $\mathcal{T}$. Let $t = |T_0|$, $\ell = \Sigma_{r=1}^{m} |T_r|$ and $\alpha_r = \Sigma_{i=1}^{r-1} |T_i|$ for $r = 1, \ldots, m$. We consider the isomorphic copy $(\mathbb{N}; \leq_{\mathcal{T}}) \cong$ UF(Γ) where $T_0 \mapsto \{0, \ldots, |T_0|\}$ and the j^{th} copy of $\mathcal{T}_r$ maps to $\{t + (j-1)\ell + \alpha_r, \ldots, t + (j-1)\ell + \alpha_{r+1} - 1\}$. The appropriate unary automaton will have parameters t, ℓ. Each q_j on the $(1,1)$-tail has $(\diamond, 1)$- and $(1, \diamond)$-tails of length t, and a $(\diamond, 1)$-loop of length ℓ. Each q_j on the $(1,1)$-loop has a $(\diamond, 1)$-tail and $(\diamond, 1)$-loop, each of length ℓ. All $(1,1)$-states are in F. Let $\varphi_0 : T_0 \to \{0, \ldots, t-1\}$ and $\varphi_r : T_r \to \{t + \alpha_r, \ldots, t + \alpha_{r+1} - 1\}$ be isomorphisms that preserve the tree order. We use φ_0 to specify which $(1, \diamond)$-and $(\diamond, 1)$-tail states from the $(1,1)$-tail are accepting. Similarly, we use $\varphi_1, \ldots, \varphi_m$ and σ, X from the parameter set to specify those state in $(\diamond, 1)$-loops off the $(1,1)$-tail and in $(\diamond, 1)$-tails and loops off the $(1,1)$-loop that are accepting. Then $(\mathbb{N}; L(\mathcal{A})) \cong$ UF(Γ). ∎

4.5.2 An efficient solution to the isomorphism problem

We wish to use the characterization of unary automatic trees to solve the isomorphism problem. However, two tree-unfoldings may be isomorphic even if the associated parameter sets are not isomorphic term-by-term. For example, if $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ is any parameter set and $\Gamma' = (\mathcal{T}_0', \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma', X)$ where $\mathcal{T}_0'$ is the subtree of UF(Γ) containing one copy of each $\mathcal{T}_0, \ldots, \mathcal{T}_m$ and σ' is obtained from σ by setting $\sigma(i) = X(i)$ if $X(i) \neq \emptyset$, then UF(Γ) $\cong$ UF(Γ'). In previous section, we obtained canonical isomorphism invariants: $\alpha_{\mathcal{L}}$ for linear orders and $h_{\mathcal{E}}$ for equivalence structures. We now define an analogue for trees. Fix a computable linear order $\preceq$ on the set of finite trees.

Definition 4.5.7 *The* canonical parameter set *of a unary automatic tree $\mathcal{T} = (\mathbb{N}; \leq_{\mathcal{T}})$ is the parameter set $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ such that* UF(Γ) $\cong \mathcal{T}$ *and which is minimal in the following sense:*

1. *As finite trees, $T_1 \preceq \ldots \preceq T_m$.*
2. *If $T_i \cong T_j$, $\sigma(i) = \sigma(j)$, and $X(i) = X(j) = \emptyset$ then $i = j$.*
3. *Each T_i $(1 \leq i \leq m)$ is minimal: If $X(i) \neq \emptyset$ then if $y_1 \leq_{\mathcal{T}} y_2 \leq_{\mathcal{T}} X(i)$ the subtree with domain $\{z \mid y_1 \leq_{\mathcal{T}} z \wedge y_2 \not\leq_T z\}$ is not isomorphic to the subtree with domain $\{z \mid y_2 \leq_{\mathcal{T}} z \wedge X(i) \not\leq_T z\}$.*
4. *T_0 is minimal: T_0 has the fewest possible nodes and for all $i \in \{1, \ldots, m\}$ where $X(i) \neq \emptyset$, there is no $y \in T_0$ such that $y \leq_{\mathcal{T}} \sigma(i)$ and the subtree with domain $\{z \mid y \leq_{\mathcal{T}} z \wedge \sigma(i) \not\leq_T z\}$ is isomorphic to T_i.*

Lemma 4.5.8 *Suppose $\mathcal{T}, \mathcal{T}'$ are unary automatic trees with canonical parameter sets Γ, Γ'. Then, $\mathcal{T} \cong \mathcal{T}'$ if and only if Γ, Γ' have the same number (m) of finite trees, $(\mathcal{T}_0, \sigma) \cong (\mathcal{T}_0', \sigma')$, and for $1 \leq i \leq m$, $(\mathcal{T}_i, X(i)) \cong (\mathcal{T}_i', X'(i))$.*

Proof. It is easy to see that if $\mathcal{T}$ and $\mathcal{T}'$ have term-by-term isomorphic canonical parameter sets they are isomorphic. Conversely, suppose $\mathcal{T} \cong \mathcal{T}'$ and their canonical parameter sets are $(\mathcal{T}_0, \ldots, \mathcal{T}_{m_1}, \sigma, X)$ and $(\mathcal{T}'_0, \ldots, \mathcal{T}'_{m_2}, \sigma', X')$, respectively. Each infinite subtree of the form $(\{y \mid \sigma(i) \leq y\}; \leq_{\mathcal{T}})$, $1 \leq i \leq m$, which contains infinitely many copies of $\mathcal{T}_i$, embeds into a subtree of $\mathcal{T}'$. By (2) in Definition 4.5.7, $m_1 = m_2$. By the minimality condition on $\mathcal{T}_i, \mathcal{T}'_i$ and by the ordering of the finite trees in each parameter set, the subtree of $\mathcal{T}$ containing infinitely many copies of $\mathcal{T}_i$ can embed into the subtree of $\mathcal{T}'$ containing infinitely many copies of $\mathcal{T}'_i$ for all $i \in \{1, \ldots, m_1\}$ and vice versa. Similarly for $\mathcal{T}_i, \mathcal{T}'_{i'}$ such that $X(i) = X'(i') = \emptyset$. By minimality of $\mathcal{T}_0, \mathcal{T}'_0$, $\forall 1 \leq i \leq m_1$ $(\mathcal{T}_i, X(i)) \cong (\mathcal{T}'_i, X'(i))$. Let t_i be the root of the first copy of $\mathcal{T}_i$ in $\mathcal{T}$ and t'_i be the root of the first copy of $\mathcal{T}'_i$ in $\mathcal{T}'$.

$$\begin{aligned}(\mathcal{T}_0, \sigma) &\cong (\{y \mid y \in T_0 \wedge \forall i \in \{1, \ldots, m\} : \neg t_i \leq_T y\}; \leq_T) \\ &\cong (\{y \mid y \in T'_0 \wedge \forall i \in \{1, \ldots, m\} : \neg t'_i \leq_{T'} y\}; \leq_{T'}) \cong (\mathcal{T}'_0, \sigma')\end{aligned}$$

∎

We can now use the canonical parameter set to define an $(\mathsf{FO} + \exists^\infty)$-formula $\varphi_{\mathcal{T}}$ that describes the isomorphism type of $\mathcal{T}$ (as in Corollaries 4.3.4 and 4.4.3). This is sufficient for proving decidability of the isomorphism problem for unary automatic trees. We now show that the isomorphism problem is decidable in polynomial time.

Theorem 4.5.9 *The isomorphism problem for unary automatic trees is decidable in time $O(n^4)$ in the sizes of the input automata.*

Suppose we can compute the canonical parameter set of a tree from a unary automaton. Given two unary automatic trees, we could use Lemma 4.5.8 and a decision procedure for isomorphism on finite trees to solve the isomorphism problem on unary automatic trees.

Lemma 4.5.10 *If $\leq_{\mathcal{T}}$ is recognized by unary automaton with n states, there is an $O(n^4)$ time algorithm that computes the canonical parameter set of $\mathcal{T}$*

Proof. We divide the construction into two pieces: first compute a parameter set Γ where $\mathcal{T} \cong \mathrm{UF}(\Gamma)$, then "minimize it", i.e., compute the canonical parameter set from Γ. Recall the proof that any unary automaton has an associated parameter set (from the proof of Theorem 4.5.4). Computing the sets A and C requires searching for the appropriate accepting states on the $(\diamond, 1)$-tail and loop out of each state on the $(1, 1)$-loop. For each $j \in \{t, \ldots, t + \ell - 1\}$, let ℓ_j be the length of $(\diamond, 1)$-loop out of q_j, and $\tilde{t}_j$ be the sum of the lengths of the $(\diamond, 1)$-tail and the $(1, \diamond)$-tail out of q_j. We may determine both n_j and the class $[j]_\sim$ by checking at most ℓ_j many states on the $(\diamond, 1)$-loop and $\tilde{t}_j$ other states. In all, this takes time $O\left(\sum_{j=t}^{t+\ell-1} (\ell_j + \tilde{t}_j)\right)$

Suppose $[j]_\sim$ represents finitely many components in $\mathcal{T}$. Each component is obtained by unfolding a finite tree $\mathcal{T}'$ of size $|[j]_\sim|$ on some $x \in T'$. The tree order $\leq_{T'}$ can be computed by reading all the $(\diamond, 1)$- and $(1, \diamond)$-states out of each q_k where $k \sim j$. The node $x \in T'$ is the $\leq_{T'}$-maximal node that is in some $(k + i\ell)_{i\in\mathbb{N}}$ with $k \in C$. Again, the number of states out of q_j that need to be read is ℓ_j and computing $\mathcal{T}'$ takes time $O(\sum_{j=t}^{t+\ell-1} \ell_j + \tilde{t}_j)$. We need n_j isomorphic copies of $\mathcal{T}'$ in Γ, a total of $O(n_j|[j]_\sim|)$ nodes. Thus, to define all $\mathcal{T}'$ in the parameter set corresponding to these $\sim$-equivalence classes takes time $O(n^2)$.

On the other hand, $[j]_\sim$ may represent infinitely many pairwise isomorphic independent trees, each of which contains $|[j]_\sim|$ nodes. To compute $\mathcal{T}'$ isomorphic to these independent trees, we read the $(\diamond, 1)$- and the $(1, \diamond)$-tail out of each q_k with $k \sim j$. This takes time $O(n)$. We call a node $x \in \{0, \ldots, t-1\}$ a *parent* of $[j]_\sim$ if it is the immediate ancestor of infinitely many trees represented by $[j]_\sim$. If $[j]_\sim$ has c parents then there will be c copies of $\mathcal{T}'$ in the parameter set we are building, each of which has $X(i) = \emptyset$ and with different values of $\sigma(i)$.

Claim. There is an algorithm that runs in time $O(n^4)$ and computes all parents of $\sim$-equivalence classes representing independent trees in $\mathcal{T}$.

Proof of claim. Suppose $[j]_\sim$ represents infinitely many independent trees whose roots are from $(j + i\ell)_{i\in\mathbb{N}}$. For each $k \in \{0, \ldots, t-1\}$, let t_k be the length of the $(\diamond, 1)$-tail out of q_k and ℓ_k be the length of the $(\diamond, 1)$-loop out of q_k. We describe an algorithm that computes the parents of $[j]_\sim$. The algorithm processes the subtree of $\mathcal{T}$ restricted to $\{0, \ldots, t-1\}$, beginning at the leaves and moving downwards (we process a node only after all of its children have been processed). For each node k we determine whether it is a parent of $[j]_\sim$.

- *Case 1.* If k is a leaf, we search for $i \in \{t_k, \ldots, t_k + \ell_k - 1\}$ such that $\Delta(q_k, (\diamond, 1)^{i\ell+j-k}) \in F$. We can find such an i if and only if there are infinitely many independent trees associated to $[j]_\sim$ descending from k in $\mathcal{T}$.

- *Case 2.* If k is an internal node but has no children which are parents of $[j]_\sim$, process it as though it were a leaf. Otherwise, let $k_1, \ldots, k_r$ be children of k which are parents of $[j]_\sim$. Let U_i, V_i, D_i be subsets of $\{t_{k_i}, \ldots, t_{k_i} + \ell_{k_i}\}$ defined as

$$U_i = \{x \mid \Delta(q_k, (\diamond, 1)^{x\ell+j-k}) \in F\}, \qquad V_i = \{x \mid \Delta(q_{k_i}, (\diamond, 1)^{x\ell+j-k_i}) \in F\},$$

  and $D_i = U_i - V_i$. Let $\ell' = \max\{\ell_{k_1}, \ldots, \ell_{k_r}\}$ and let $D'_i = \{x + i\ell_k \mid x \in D_i \wedge x + i\ell_k < \ell'\ell_k\}$. Then k is a parent of $[j]_\sim$ if and only if $D'_1 \cap \cdots \cap D'_r \neq \emptyset$.

Correctness. In Case 1, if there is no $i' \geq t_k$ such that $\Delta(q_k, (\diamond, 1)^{i'\ell+j-k}) \in F$, there are only finitely many independent trees represented by $[j]_\sim$ descending from k. Moreover, if such an i' exists, it must be on the $(\diamond, 1)$-loop off q_k and so we can stop looking for it after we have checked all ℓ_k states. For Case 2, note that $x \in D'_1 \cap \cdots \cap D'_r$ if and only if k is the immediate ancestor for all nodes in $\{j + (x + i\ell'\ell_k)\ell\}_{i\in\mathbb{N}}$, if and only if k is the immediate

ancestor for some node in $\{j + (x + i\ell'\ell_k)\ell\}_{i\in\mathbb{N}}$. Therefore if $D'_1 \cap \cdots \cap D'_r \neq \emptyset$, then k is a parent of $[j]_\sim$. Suppose $D'_1 \cap \cdots \cap D'_r = \emptyset$ and k is a parent of $[j]_\sim$. Then k is the immediate ancestor for $\{j + (s + im)\ell\}_{i\in\mathbb{N}}$ for some $m > \ell'\ell_k$ and $s < m$. If $s < \ell'\ell_k$, then $s \in D'_1 \cap \cdots \cap D'_r$. Therefore $s \geq \ell'\ell_k$. Say $s = s' + i\ell'\ell_k$ where $s' < \ell'\ell_k$. Then k is the immediate ancestor of $j + (s + \ell'\ell_k m)\ell = j + (s' + (m + i)\ell\ell_k)\ell$. Therefore $s' \in D'_1 \cap \cdots \cap D'_r$. Contradiction. Hence the algorithm is correct.

Complexity. Checking if a leaf k is a parent for $[j]_\sim$ takes time $O(\ell_k\ell)$. When k is an internal node, computing U_i and V_i takes time $O(\ell_k\ell_{k_i}\ell)$. The size of each U_i and V_i is bounded by ℓ_{k_i}, therefore computing D_i takes $O(\ell_{k_i})$. Computing each D'_i takes time $O(\ell'\ell_k)$. We need to carry out the above operations at most t times (at most once for each node in $\{0, \ldots, t-1\}$). Therefore, the algorithm takes time $O(t\hat{\ell}^2\ell)$, where $\hat{\ell}$ is the maximal $(\diamond, 1)$-loop length out of all q_k, $k \in \{0, \ldots, t-1\}$. We iterate the intersection operation r times to compute the intersection of all the D'_i's; therefore, we perform a total of at most t intersection operations, each taking time $O(\hat{\ell}^2)$. Since $\hat{\ell} < n$, the algorithm takes time $O(n^4)$. We can run the above algorithm simultaneously for all equivalence classes $[j]_\sim$ representing independent trees without increasing the time complexity.

With the above claim in hand, we can resume our construction of the parameter set Γ. The finite tree $\mathcal{T}_0$ in Γ contains all nodes in $\{0, \ldots, t-1\}$ and finitely many independent trees. Deciding which independent trees to put into $\mathcal{T}_0$ uses the claim and therefore takes $O(n^4)$. Computing the tree order $\leq_{T_0}$ on $\{0, \ldots, t-1\}$ requires reading the $(\diamond, 1)$- and $(1, \diamond)$-tail out of each q_k $(0 \leq k < t)$ at most once. This steps again takes time $O(n)$. Thus, in time $O(n^4)$ time, we have computed $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ such that $\mathcal{T} \cong \mathrm{UF}(\Gamma)$. Since nodes in $\mathcal{T}_0$ can be parents to more than one anti-chain, $m \leq t\ell + \sum_{j\in C} n_j \leq t\ell + n$.

We now use Γ to obtain a canonical parameter set for $\mathcal{T}$. For each $i \in \{1, \ldots, m\}$ with $X(i) \neq \emptyset$, look for $y_1, y_2 \in T_i$ such that $y_1 <_{\mathcal{T}} y_2 <_{\mathcal{T}} X(i)$, and the subtree of T_i with domain $\{z \mid y_1 \leq_{\mathcal{T}} z \wedge y_2 \not\leq_{\mathcal{T}} z\}$ is isomorphic to the subtree with domain $\{z \mid y_2 \leq_{\mathcal{T}} z \wedge X(i) \not\leq_{\mathcal{T}} z\}$. If such y_1, y_2 exist, remove all $z \geq_{T_i} y_1$ from T_i. For each $i, j, 1 \leq i < j \leq m$, such that $X(i) = X(j) = \emptyset$, if $T_i \cong T_j$ and $\sigma(i) = \sigma(j)$ then remove T_j. Thus, each T_i satisfies the minimality condition for the canonical parameter set. Since the isomorphism problem for finite trees is decidable in linear time [58], this step can be done in time $O(\sum_{i=1}^m |T_i|^2)$.

For each $i \in \{1, \ldots, m\}$ with $X(i) \neq \emptyset$, let t_i be the root of $T_i \times \{0\}$. Look for $x \in T_0$ such that $x \leq_{\mathcal{T}} \sigma(i)$, and the subtree of T_0 with domain $\{y \mid x \leq_{\mathcal{T}} y \wedge t_i \not\leq_T y\}$ is isomorphic to T_i. If such an x exists, remove all $y \geq_{T_0} x$ from T_0. Now T_0 satisfies the minimality condition. Again this step can be done in time $O(\sum_{i=1}^m |T_i|^2)$.

For each $i \in \{1, \ldots, m\}$, search for the $<_{T_0}$-least x such that the subtree of T_0 with domain $\{z \in T_0 \mid x \leq_{T_0} z\}$ is isomorphic to a subtree of T_i with domain $\{z \in T_i \mid y \leq_{T_i} z\}$ for some $y <_{T_i} X(i)$. If such an x exists, remove all $y \geq_{T_0} x$ from T_0. This step ensures that T_0 has the fewest possible nodes with respect to T_i; it can be done in time $O(\sum_{i=1}^m |T_i|^2)$.

Finally, we permute $T_1, \ldots, T_m$ so that $T_1 \preceq \ldots \preceq T_m$. We assume that finite trees can be efficiently encoded as natural numbers and hence applying a sorting algorithm on m of them takes $O(m \log m)$. Whenever we find $T_i \cong T_j (i \neq j)$ with $\sigma(i) = \sigma(j)$ and $X(i) = X(j) = \emptyset$, keep T_i and delete T_j. Converting Γ to a canonical parameter set takes $O(n^3)$ and thus we have built such a canonical parameter set in $O(n^4)$ time. ∎

Proof of Theorem 4.5.9. Suppose $\mathcal{T}_1, \mathcal{T}_2$ are presented by unary automata $\mathcal{A}_1, \mathcal{A}_2$ with n_1, n_2 states (respectively). Let $n = \max\{n_1, n_2\}$ By Theorem 4.5.4 and Lemma 4.5.10, deciding if $\mathcal{T}_1 \cong \mathcal{T}_2$ reduces to checking finitely many isomorphisms of finite trees. The appropriate canonical parameter sets are built in $O(n^3)$ time and each have $O(n^2)$ finite trees, each of size $O(n)$. Hence, this isomorphism algorithm runs in $O(n^3)$ time. ∎

4.5.3 State complexity

Suppose $\mathcal{T} = \mathrm{UF}(\Gamma)$ and $\Gamma = (\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_m, \sigma, X)$ is the canonical parameter set of T. Let $t = |T_0|$ and $\ell = \sum_{i=1}^{m} |T_i|$. The proof of Theorem 4.5.4 gives an upper bound on the state complexity of unary automatic trees in terms of t and ℓ.

Theorem 4.5.11 *The state complexity of unary automatic tree T is less than $(t + \ell)^2 - t\ell + t + \ell$ and greater than ℓ^2.*

Proof. The automaton $\mathcal{A}$ built in the proof of Theorem 4.5.4 has size $t(t + \ell) + 2\ell^2 + t + \ell$. To further reduce the number of states, we can permute the domain of the tree so that if $j \in A$ then q_j has a $(\diamond, 1)$-tail of length ℓ and a $(\diamond, 1)$-loop of length 1, and if $j \in C$ then q_j has a $(\diamond, 1)$-loop of length ℓ and no $(\diamond, 1)$-tail. Therefore the size of $\mathcal{A}$ is $t(t+\ell)+\ell^2+t+\ell = (t+\ell)^2-t\ell+t+\ell$.

When $\mathcal{T}_1, \ldots, \mathcal{T}_m$ are pairwise non-isomorphic, the loop length of $\mathcal{A}$ is at least ℓ and there are at least ℓ $(\diamond, 1)$-states out of each q_j on the $(1, 1)$-loop. Therefore the state complexity is bounded below by ℓ^2. ∎

Corollary 4.5.12 *The (unary) state complexity of a unary automatic tree $\mathcal{T}$ is quadratic in the parameters t, ℓ of its canonical parameter set.*

Chapter 5

The Isomorphism Problem for Automatic Structures

This chapter continues to study automatic structures over arbitrary finite alphabets. It is well-known that the isomorphism problem for automatic structures in general is complete for Σ^1_1, the first existential level of the analytical hierarchy [72]. Using direct interpretations, the isomorphism problem is shown to be also Σ^1_1-complete for the class of automatic successor trees, automatic undirected graphs, automatic partial orders etc. On the other hand, the problem is decidable for automatic ordinals and Boolean algebras. An intermediate class it the class of automatic graphs of finite degrees, for which the isomorphism problem is complete for Π^0_3.

In [71], B. Khoussainov and A. Nerode asked, among other open questions on automatic structures, whether the isomorphism problem is decidable for automatic equivalence structures (Question 4.2 of [71]) and automatic linear orders (Question 4.3 of [71]). These questions have been open for more than 15 years. For the class of equivalence structures, it has been conjectured that the isomorphism problem is decidable [71]. Also presented in [71] are the following questions:

- (Question 4.6 of [71]) Provide natural examples of classes of automatic structures for which the isomorphism is Σ^0_k- or Π^0_k-complete, where $k \in \mathbb{N}$.
- (Question 4.7 of [71]) Prove that between any word automatic isomorphic linear orders (trees) there is always a computable isomorphism.

In this chapter, we answer all the above questions negatively (with the exception of Ques.4.6) by proving the following:

– The isomorphism problem for automatic equivalence structures is Π^0_1-complete.

– The isomorphism problem for automatic successor trees of finite height $k \geq 2$ is Π^0_{2k-3}-complete.

– The isomorphism problem for automatic linear orders is hard for every level of the arithmetic hierarchy.

– For any $k \in \mathbb{N}$, there exist two isomorphic automatic trees of finite height (and two automatic linear orders) without any Σ^0_k-isomorphism.

Most hardness proofs for automatic structures, in particular the Σ^1_1-hardness proof for the isomorphism problem of automatic structures from [72], use configuration graphs of Turing-machines (which are automatic structures as stated in Example 2.5.8). This technique does not generalize to transitive relations (the transitive closure of the configuration graph of a Turing-machine cannot be automatic in general), and hence it cannot be applied to automatic equivalence structures and linear orders. Our proofs are based on the undecidability of Hilbert's 10^{th} problem (see Example 2.3.5). The technique is used in Honkala [56] in proving the undecidability of whether a rational power series has range $\mathbb{N}$. Using a similar encoding, we show that the isomorphism problem for automatic equivalence relation is Π^0_1-complete. Next, we extend our technique to show that the isomorphism problem for automatic successor trees of height $k \geq 2$ is Π^0_{2k-3}-complete. In some sense, our result for equivalence relations makes up the induction base $k = 2$. Finally, we solve the problem for linear orders using a more elaborate construction involving shuffle sums.

5.1 Automatic equivalence structures

In this section, we prove that the isomorphism problem for automatic equivalence structures is Π^0_1-complete. This result can be also deduced from our result for automatic trees (Section 5.2). But the case of equivalence structures is a good starting point for introducing our techniques.

This chapter uses the following operations on automata:

– Given two automata $\mathcal{A}_1 = (S_1, \Delta_1, I_1, F_1)$ and $\mathcal{A}_2 = (S_2, \Delta_2, I_2, F_2)$ over the same alphabet Σ, we use $\mathcal{A}_1 \uplus \mathcal{A}_2$ to denote the automaton obtained by taking the disjoint union of $\mathcal{A}_1$ and $\mathcal{A}_2$. Note that for any word $u \in \Sigma^+$, the number of accepting runs of $\mathcal{A}_1 \uplus \mathcal{A}_2$ on u is equal to the sum of the numbers of accepting runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ on u.

– We use $\mathcal{A}_1 \times \mathcal{A}_2$ to denote the Cartesian product of $\mathcal{A}_1$ and $\mathcal{A}_2$. It is the automaton $(S_1 \times S_2, \Delta, I_1 \times I_2, F_1 \times F_2)$, where

$$\Delta = \left\{((p_1, p_2), \sigma, (q_1, q_2)) \mid (p_1, \sigma, q_1) \in \Delta_1, (p_2, \sigma, q_2) \in \Delta_2\right\}$$

Then, clearly, the number of accepting runs of $\mathcal{A}_1 \times \mathcal{A}_2$ on a word $u \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ is the product of the numbers of accepting runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ on u. In particular, if $\mathcal{A}_1$ is deterministic, then the number of accepting runs of $\mathcal{A}_1 \times \mathcal{A}_2$ is the same as the number of accepting runs of $\mathcal{A}_2$ on u.

– If $\mathcal{A}$ is a non-deterministic automaton and D is a regular language, we write $D \uplus \mathcal{A}$ (resp. $D \cap \mathcal{A}$) for the automaton $\mathcal{A}_D \uplus \mathcal{A}$ (resp. $\mathcal{A}_D \times \mathcal{A}$), where $\mathcal{A}_D$ is some deterministic automaton for the language D.

Lemma 5.1.1 *The isomorphism problem for automatic equivalence structures is in* Π^0_1.

Proof. Let $\mathcal{E}$ be an automatic equivalence structure. Recall that $h_{\mathcal{E}}$ is the height function of $\mathcal{E}$ (defined in Section 4.4). Note that for given $n \in \mathbb{N} \cup \{\aleph_0\}$, the value $h_{\mathcal{E}}(n)$ can be computed effectively: one can define in $\mathsf{FO} + \exists^\infty$ the set of all $\leq_{\text{llex}}$-least elements that belong to an equivalence class of size n.

Given two automatic equivalence structures $\mathcal{E}_1 = (D_1; E_1)$ and $\mathcal{E}_2 = (D_2; E_2)$, deciding if $\mathcal{E}_1 \cong \mathcal{E}_2$ amounts to checking if $h_{\mathcal{E}_1} = h_{\mathcal{E}_2}$:

$$\forall n \in \mathbb{N} : h_{\mathcal{E}_1}(n) = h_{\mathcal{E}_2}(n).$$

Therefore, the isomorphism problem for automatic equivalence structures is in Π^0_1. ∎

For the Π^0_1 lower bound, we use a reduction from Hilbert's 10^{th} problem: Given a polynomial $p(x_1, \ldots, x_k) \in \mathbb{Z}[x_1, \ldots, x_k]$, decide whether the equation $p(x_1, \ldots, x_k) = 0$ has a solution in $\mathbb{N}_+$. Example 2.3.5 shows that the following set is Π^0_1-complete:

$$\{(p_1(\overline{x}), p_2(\overline{x})) \in \mathbb{N}[x_1, \ldots, x_k]^2 \mid \forall \overline{c} \in \mathbb{N}_+^k : p_1(\overline{c}) \neq p_2(\overline{c})\}.$$

For a symbol a, let Σ_k^a denote the alphabet

$$\Sigma_k^a = \{a, \diamond\}^k \setminus \{(\diamond, \ldots, \diamond)\}$$

and let σ_i denote the i^{th} component of $\sigma \in \Sigma_k^a$. For $\overline{e} = (e_1, \ldots, e_k) \in \mathbb{N}_+^k$, write $a^{\overline{e}}$ for the word

$$a^{e_1} \otimes a^{e_2} \otimes \cdots \otimes a^{e_k} .$$

For a language L, we write $\otimes_k(L)$ for the language

$$\{u_1 \otimes u_2 \otimes \cdots \otimes u_k \mid u_1, \ldots, u_k \in L\}.$$

Lemma 5.1.2 *There exists an algorithm that, given a non-zero polynomial* $p(\overline{x}) \in \mathbb{N}[\overline{x}]$ *in* k *variables, constructs a non-deterministic automaton* $\mathcal{A}[p(\overline{x})]$ *on the alphabet* Σ_k^a *with* $L(\mathcal{A}[p(\overline{x})]) = \otimes_k(a^+)$ *such that for all* $\overline{c} \in \mathbb{N}_+^k$: $\mathcal{A}[p(\overline{x})]$ *has exactly* $p(\overline{c})$ *accepting runs on input* $a^{\overline{c}}$.

Proof. The automaton $\mathcal{A}[p(\overline{x})]$ is build by induction on the construction of the polynomial p, the base case is provided by the polynomials 1 and x_i.

Let $\mathcal{A}[1]$ be a deterministic automata accepting $\otimes_k(a^+)$. Next, suppose $p(x_1, \ldots, x_k) = x_i$ for some $i \in \{1, \ldots, k\}$. Let $S = \{q_1, q_2\}$, $I = \{q_1\}$ and $F = \{q_2\}$. Define Δ as

$$\Delta = \{(q_1, \sigma, q_j) \mid j \in \{1, 2\}, \sigma \in \Sigma_k^a, \sigma_i = a\} \cup \{(q_2, \sigma, q_2) \mid \sigma \in \Sigma_k^a\}.$$

When the automaton $\mathcal{A}[p(\overline{x})] = (S, I, \Delta, F)$ runs on an input word $a^{\overline{c}}$, it has exactly c_i many times the chance to move from state q_1 to the final state q_2. Therefore there are exactly $c_i = p(\overline{c})$ many accepting runs on $a^{\overline{c}}$.

Let $p_1(\overline{x})$ and $p_2(\overline{x})$ be polynomials in $\mathbb{N}[\overline{x}]$. Assume as inductive hypothesis that there are two automata $\mathcal{A}[p_1(\overline{x})]$ and $\mathcal{A}[p_2(\overline{x})]$ such that for $i \in \{1, 2\}$ the number of accepting runs of $\mathcal{A}[p_i(\overline{x})]$ on $a^{\overline{c}}$ equals $p_i(\overline{c})$.

For $p(\overline{x}) = p_1(\overline{x}) + p_2(\overline{x})$, set $\mathcal{A}[p(\overline{x})] = \mathcal{A}[p_1(\overline{x})] \uplus \mathcal{A}[p_2(\overline{x})]$. Then, the number of accepting runs of $\mathcal{A}[p(\overline{x})]$ on $a^{\overline{c}}$ is $p_1(\overline{c}) + p_2(\overline{c})$.

For $p(\overline{x}) = p_1(\overline{x}) \cdot p_2(\overline{x})$, let $\mathcal{A}[p(\overline{x})] = \mathcal{A}[p_1(\overline{x})] \times \mathcal{A}[p_2(\overline{x})]$. Then, the number of accepting runs of $\mathcal{A}[p(\overline{x})]$ on $a^{\overline{c}}$ is $p_1(\overline{c}) \cdot p_2(\overline{c})$. ∎

Let $\mathcal{A} = (S, I, \Delta, F)$ be a non-deterministic finite automaton with alphabet Σ. We define an automaton $\mathrm{Run}_{\mathcal{A}} = (S, I, \Delta', F)$ with alphabet Δ and

$$\Delta' = \{(p, (p, a, q), q) \mid (p, a, q) \in \Delta\}.$$

Let $\pi : \Delta^* \to \Sigma^*$ be the projection morphism with $\pi(p, a, q) = a$. The following lemma is immediate from the definition.

Lemma 5.1.3 *For $u \in \Delta^+$ we have: $u \in L(\mathrm{Run}_{\mathcal{A}})$ if and only if u forms an accepting run of $\mathcal{A}$ on $\pi(u)$ (which in particular implies $\pi(u) \in L(\mathcal{A})$).*

This lemma implies that for all words $w \in \Sigma^+$, $|\pi^{-1}(w) \cap L(\mathrm{Run}_{\mathcal{A}})|$ equals the number of accepting runs of $\mathcal{A}$ on w. Note that this does not hold for $w = \varepsilon$.

Consider a non-zero polynomial $p(\overline{x}) \in \mathbb{N}[x_1, \ldots, x_k]$. Let the automaton $\mathcal{A} = \mathcal{A}[p(\overline{x})]$ satisfy the properties guaranteed by Lemma 5.1.2 and let $\mathrm{Run}_{\mathcal{A}}$ be as defined above. Define an automatic equivalence structure $\mathcal{E}(p)$ whose domain is $L(\mathrm{Run}_{\mathcal{A}}) \setminus \{\varepsilon\}$. Moreover, two words $u, v \in L(\mathrm{Run}_{\mathcal{A}}) \setminus \{\varepsilon\}$ are equivalent if and only if $\pi(u) = \pi(v)$. By definition and Lemma 5.1.2, a natural number $y \in \mathbb{N}_+$ belongs to $\mathrm{Img}_+(p)$ if and only if there exists a word $u \in L(\mathcal{A})$ with precisely y accepting runs, if and only if $\mathcal{E}(p)$ contains an equivalence class of size y.

It is well known that the function $C : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ with

$$C(x, y) = (x + y)^2 + 3x + y \tag{5.1}$$

is injective ($C(x,y)/2$ defines a pairing function, see e.g. [56]). In the following, let $\mathcal{E}_{\text{Good}}$ denote the countably infinite equivalence structure with

$$h_{\mathcal{E}_{\text{Good}}}(n) = \begin{cases} \infty & \text{if } n \in \{C(y,z) \mid y,z \in \mathbb{N}_+, y \neq z\} \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 5.1.4 *The set of automatic presentations $\mathcal{P}$ with $\mathcal{S}(\mathcal{P}) \cong \mathcal{E}_{Good}$ is hard for Π^0_1.*

Proof. For non-zero polynomials $p_1(\overline{x}), p_2(\overline{x}) \in \mathbb{N}[x_1, \ldots, x_k]$, define the following three (non-zero) polynomials from $\mathbb{N}[x_1, \ldots, x_k]$ (with $k \geq 2$):

$$S_1(\overline{x}) = C(p_1(\overline{x}), p_2(\overline{x})), \qquad S_2(\overline{x}) = C(x_1 + x_2, x_1), \qquad S_3(\overline{x}) = C(x_1, x_1 + x_2).$$

Let $\mathcal{E}(S_1)$, $\mathcal{E}(S_2)$, and $\mathcal{E}(S_3)$ be the automatic equivalence structures corresponding to these polynomials according to the above definition. Finally, let $\mathcal{E}$ be the disjoint union of $\aleph_0$ many copies of these three equivalence structures.

If $p_1(\overline{c}) = p_2(\overline{c})$ for some $\overline{c} \in \mathbb{N}_+^k$, then there is $y \in \mathbb{N}_+$ such that $C(y,y) \in \text{Img}_+(S_1)$. Therefore in $\mathcal{E}$ there is an equivalence class of size $C(y,y)$ and no such equivalence class exists in $\mathcal{E}_{\text{Good}}$. Hence $\mathcal{E} \ncong \mathcal{E}_{\text{Good}}$.

Conversely, suppose that $p_1(\overline{c}) \neq p_2(\overline{c})$ for all $\overline{c} \in \mathbb{N}_+^k$. For all $y,z \in \mathbb{N}_+$, $\mathcal{E}$ contains an equivalence class of size $C(y,z)$ if and only if $C(y,z)$ belongs to $\text{Img}_+(S_1) \cup \text{Img}_+(S_2) \cup \text{Img}_+(S_3)$, if and only if $y \neq z$, if and only if $\mathcal{E}_{\text{Good}}$ contains an equivalence class of size $C(y,z)$. Therefore, for any $s \in \mathbb{N}_+$, $\mathcal{E}$ contains an equivalence class of size s if and only if $\mathcal{E}_{\text{Good}}$ contains an equivalence class of size s. Hence $\mathcal{E} \cong \mathcal{E}_{\text{Good}}$.

In summary, we have reduced the Π^0_1-hard problem

$$\{(p_1(\overline{x}), p_2(\overline{x})) \in \mathbb{N}[x_1, \ldots, x_k]^2 \mid k \geq 2, \forall \overline{c} \in \mathbb{N}_+^k : p_1(\overline{c}) \neq p_2(\overline{c})\}$$

to the set of automatic presentations of $\mathcal{E}_{\text{Good}}$. Hence the proposition is proved. ∎

Theorem 5.1.5 *The isomorphism problem for automatic equivalence structures is Π^0_1-complete.*

Proof. At the beginning of this section, we already argued that the isomorphism problem is in Π^0_1; hardness follows immediately from Proposition 5.1.4, since $\mathcal{E}_{\text{Good}}$ is necessarily automatic. ∎

Definition 5.1.6 *A graph $\mathcal{G}$ is* strongly locally finite *if every component of $\mathcal{G}$ forms a finite graph.*

Corollary 5.1.7 *The isomorphism problem for automatic strongly locally finite graphs is Π^0_1-complete.*

Proof. The Π^0_1-hardness immediately follows from Theorem 5.1.5 since the equivalence structure $\mathcal{E}_{\text{Good}}$ we constructed when viewed as a graph is strongly locally finite. The Π^0_1-membership can be proved in a similar way as Lemma 5.1.1. Let $\mathcal{G}$ be an automatic strongly locally finite graph. For every finite graph $\mathcal{H}$, define $h_{\mathcal{G}}(\mathcal{H})$ as the (possibly infinite) number of nodes in $\mathcal{G}$ whose component is isomorphic to $\mathcal{H}$. Since the isomorphism type of any finite graph can be defined in first-order logic, the function $h_{\mathcal{G}}$ is computable. Note that for two automatic strongly locally finite graphs $\mathcal{G}_1$ and $\mathcal{G}_2$, $\mathcal{G}_1 \cong \mathcal{G}_2$ if and only if

$$\forall \text{ finite graph } \mathcal{H} : h_{\mathcal{G}_1}(\mathcal{H}) = h_{\mathcal{G}_2}(\mathcal{H}).$$

Hence the isomorphism problem is in Π^0_1. ∎

Remark. There exists a computable isomorphism between any two automatic presentations of a strongly locally finite graph. Indeed, let $\mathcal{G}_1 \cong \mathcal{G}_2$ to two such automatic presentations. For each node u in $\mathcal{G}_1$, we can effectively compute the component $C(u)$ of u and locate in $\mathcal{G}_2$ all copies of $C(u)$. An isomorphism can then be effectively constructed by mapping the isomorphic components in $\mathcal{G}_1$ and $\mathcal{G}_2$.

5.2 Automatic trees

In this section we assume the definition for trees as in Example 2.1.5, but will quite often refer to them as graphs for convenience (see Example 2.2.2). We use $\mathcal{T}_n$ to denote the class of automatic trees with height at most n. Let n be fixed. Then the tree order $\leq$ is FO-definable in T and this holds even uniformly for all trees from $\mathcal{T}_n$. Moreover, it is decidable whether a given automatic graph belongs to $\mathcal{T}_n$ (since the class of trees of height n can be axiomatized in first-order logic).

As a corollary to Proposition 5.1.4, we get immediately that the isomorphism problem for automatic trees of height at most 2 is undecidable:

Corollary 5.2.1 *There exists an automatic tree T_{Good} of height 2 such that the set of automatic presentations $\mathcal{P}$ with $\mathcal{S}(\mathcal{P}) \cong T_{Good}$ is Π^0_1-hard. Hence, the isomorphism problem for the class $\mathcal{T}_2$ of automatic trees of height at most 2 is Π^0_1-hard.*

Proof. Let $\mathcal{E} = (V; \equiv)$ be an automatic equivalence structure. Now build the tree $T(\mathcal{E})$ as follows:

- the set of nodes is $V \cup \{r\} \cup \{au \mid u \in V, u \text{ is } \leq_{\text{llex}}\text{-minimal in } [u]_\equiv\}$ where r and a are two new letters
- r is the root, its children are the words starting with a, and the children of au are the words from $[u]_\equiv$.

Then it is clear that $T(\mathcal{E})$ is a tree of height at most 2 and that an automatic presentation for $T(\mathcal{E})$ can be computed from one for $\mathcal{E}$. Furthermore, $\mathcal{E} \cong \mathcal{E}_{\text{Good}}$ if and only if $T(\mathcal{E}) \cong T(\mathcal{E}_{\text{Good}})$. Hence, indeed, the statement follows from Proposition 5.1.4. ∎

The hardness statement of Theorem 5.2.13 below is a generalization of this corollary to all the classes $\mathcal{T}_n$ for $n \geq 2$. But first, we prove an upper bound for the isomorphism problem for $\mathcal{T}_n$:

Proposition 5.2.2 *The isomorphism problem for the class $\mathcal{T}_n$ of automatic trees of height at most n is*

- *decidable for $n = 1$ and*
- *in Π^0_{2n-3} for all $n \geq 2$.*

Proof. We first show that $T_1 \cong T_2$ is decidable for automatic trees $T_1, T_2 \in \mathcal{T}_1$ of height at most 1: It suffices to compute the cardinality of T_i ($i \in \{1, 2\}$) which is possible since the universes of T_1 and T_2 are regular languages.

Now let $n \geq 2$ and consider $T_1, T_2 \in \mathcal{T}_n$. Let $T_i = (V_i, E_i)$, w.l.o.g. $V_1 \cap V_2 = \emptyset$, and $V = V_1 \cup V_2$, $E = E_1 \cup E_2$. For any node u in V, let $T(u)$ denote the subtree (of either T_1 or T_2) rooted at u and let $E(u)$ be the set of children of u. For $k = n-2, n-3, \ldots, 0$, we will define inductively a $\Pi^0_{2n-2k-3}$-predicate $\text{iso}_k(u_1, u_2)$ for $u_1, u_2 \in V$. This predicate expresses that $T(u_1) \cong T(u_2)$ provided u_1 and u_2 belong to level at least k. The result will follow since $T_1 \cong T_2$ if and only if $\text{iso}_0(r_1, r_2)$ holds, where r_σ is the root of T_σ.

For $k = n-2$, the trees $T(u_1)$ and $T(u_2)$ have height at most 2 and we can define $\text{iso}_{n-2}(u_1, u_2)$ as follows:

$$\forall \kappa \in \mathbb{N} \cup \{\aleph_0\}\ \forall \ell \geq 1 \left(\begin{array}{c} \exists x_1, \ldots, x_\ell \in E(u_1) : \bigwedge_{1 \leq i < j \leq \ell} x_i \neq x_j \wedge \bigwedge_{i=1}^{\ell} |E(x_i)| = \kappa \\ \iff \exists y_1, \ldots, y_\ell \in E(u_2) : \bigwedge_{1 \leq i < j \leq \ell} y_i \neq y_j \wedge \bigwedge_{i=1}^{\ell} |E(y_i)| = \kappa \end{array} \right)$$

In other words: for every $\kappa \in \mathbb{N} \cup \{\aleph_0\}$, u_1 and u_2 have the same number of children with exactly κ children. Since $\mathsf{FO} + \exists^\infty$ is uniformly decidable for automatic structures, this is indeed a Π^0_1-sentence (note that $2n - 2k - 3 = 1$ for $k = n-2$). For $0 \leq k < n-2$, we define $\text{iso}_k(u_1, u_2)$ inductively as follows:

$$\forall v \in E(u_1) \cup E(u_2)\ \forall \ell \geq 1 \left(\begin{array}{c} \exists x_1, \ldots, x_\ell \in E(u_1) : \bigwedge_{1 \leq i < j \leq \ell} x_i \neq x_j \wedge \bigwedge_{i=1}^{\ell} \text{iso}_{k+1}(v, x_i) \\ \iff \exists y_1, \ldots, y_\ell \in E(u_2) : \bigwedge_{1 \leq i < j \leq \ell} y_i \neq y_j \wedge \bigwedge_{i=1}^{\ell} \text{iso}_{k+1}(v, y_i) \end{array} \right)$$

By quantifying over all $v \in E(u_1) \cup E(u_2)$, we quantify over all isomorphism types of trees that occur as a subtree rooted at a child of u_1 or u_2. For each of these isomorphism types τ, we express that u_1 and u_2 have the same number of children x with $T(x)$ of type τ. Since by induction, $\mathrm{iso}_{k+1}(v, x_i)$ and $\mathrm{iso}_{k+1}(v, y_i)$ are $\Pi^0_{2n-2k-1}$-statements, $\mathrm{iso}_k(u_1, u_2)$ is a $\Pi^0_{2n-2k-3}$-statement. ∎

The rest of this section is devoted to proving that the isomorphism problem on the class $\mathcal{T}_n$ of automatic trees of height at most $n \geq 2$ is also Π^0_{2n-3}-hard (and therefore complete). So let $P_n(x_0)$ be a Π^0_{2n-3}-predicate. In the following lemma and its proof, all quantifiers with unspecified range, run over $\mathbb{N}_+$.

Lemma 5.2.3 *For $2 \leq i \leq n$, there are Π^0_{2i-3}-predicates $P_i(x_0, x_1, y_1, x_2, y_2, \ldots, x_{n-i}, y_{n-i})$ such that*

(i) $P_{i+1}(\overline{x})$ is logically equivalent to $\forall x_{n-i} \exists y_{n-i} : P_i(\overline{x}, x_{n-i}, y_{n-i})$ for $2 \leq i < n$ and

(ii) $\forall y_{n-i} : \neg P_i(\overline{x}, x_{n-i}, y_{n-i})$ implies $\forall x'_{n-i} \geq x_{n-i}\ \forall y_{n-i} : \neg P_i(\overline{x}, x'_{n-i}, y_{n-i})$,

where $\overline{x} = (x_0, x_1, y_1, \ldots, x_{n-i-1}, y_{n-i-1})$.

Proof. The predicates P_i are constructed by induction, starting with $i = n-1$ down to $i = 2$ where the construction of P_i does not assume that (i) or (ii) hold true for P_{i+1}.

So let $2 \leq i < n$ such that $P_{i+1}(\overline{x})$ is a $\Pi^0_{2(i+1)-3}$-predicate. Then there exists a Π^0_{2i-3}-predicate $P(\overline{x}, x_{n-i}, y_{n-i})$ such that $P_{i+1}(\overline{x})$ is logically equivalent to

$$\forall x_{n-i} \exists y_{n-i} : P(\overline{x}, x_{n-i}, y_{n-i}) .$$

But this is logically equivalent to

$$\forall x_{n-i}\ \forall x'_{n-i} \leq x_{n-i}\ \exists y_{n-i} : P(\overline{x}, x'_{n-i}, y_{n-i}) . \tag{5.2}$$

Let $\varphi(\overline{x}, x_{n-i})$ be

$$\forall x'_{n-i} \leq x_{n-i}\ \exists y_{n-i} : P(\overline{x}, x'_{n-i}, y_{n-i}) .$$

Then for any $x_{n-i} \in \mathbb{N}$,

$$\neg\varphi(\overline{x}, x_{n-i}) \implies \forall x \geq x_{n-i} : \neg\varphi(\overline{x}, x) . \tag{5.3}$$

Since $\forall x'_{n-i} \leq x_{n-i}$ is a bounded quantifier, the formula $\varphi(\overline{x}, x_{n-i})$ belongs to Σ^0_{2i-2} (see for example [109, p. 61]). Thus there is a Π^0_{2i-3}-predicate $P_i(\overline{x}, x_{n-i}, y_{n-i})$ such that

$$\varphi(\overline{x}, x_{n-i}) \iff \exists y_{n-i} : P_i(\overline{x}, x_{n-i}, y_{n-i}) . \tag{5.4}$$

Therefore (5.2) (and therefore $P_{i+1}(\overline{x})$) is logically equivalent to $\forall x_{n-i}\, \exists y_{n-i} : P_i(\overline{x}, x_{n-i}, y_{n-i})$. Moreover,

$$\begin{aligned} \forall y_{n-i} : \neg P_i(\overline{x}, x_{n-i}, y_{n-i}) &\overset{(5.4)}{\iff} \neg\varphi(\overline{x}, x_{n-i}) \\ &\overset{(5.3)}{\implies} \forall x \geq x_{n-i} : \neg\varphi(\overline{x}, x) \\ &\overset{(5.4)}{\iff} \forall x \geq x_{n-i}\, \forall y_{n-i} : \neg P_i(\overline{x}, x, y_{n-i}) \end{aligned}$$

This shows (ii). ∎

Let us fix the predicates P_i for the rest of Section 5.2. By induction on $2 \leq i \leq n$, we will construct the following trees:

- test trees $T^i_{\overline{c}} \in \mathcal{T}_i$ for $\overline{c} \in \mathbb{N}_+^{1+2(n-i)}$ (which depend on P_i) and
- trees $U^i_\kappa \in \mathcal{T}_i$ for $\kappa \in \mathbb{N}_+ \cup \{\omega\}$ (we assume the standard order on $\mathbb{N}_+ \cup \{\omega\}$).

The idea is that $T^i_{\overline{c}} \cong U^i_\kappa$ if and only if $\kappa = 1 + \inf(\{x_{n-i} \mid \forall y_{n-i} \in \mathbb{N}_+ : \neg P_i(\overline{c}, x_{n-i}, y_{n-i})\} \cup \{\omega\})$. We will not prove this equivalence, but the following simpler consequences for any $\overline{c} \in \mathbb{N}_+^{1+2(n-i)}$:

(P1) $P_i(\overline{c})$ holds if and only if $T^i_{\overline{c}} \cong U^i_\omega$.

(P2) $P_i(\overline{c})$ does not hold if and only if $T^i_{\overline{c}} \cong U^i_m$ for some $m \in \mathbb{N}_+$.

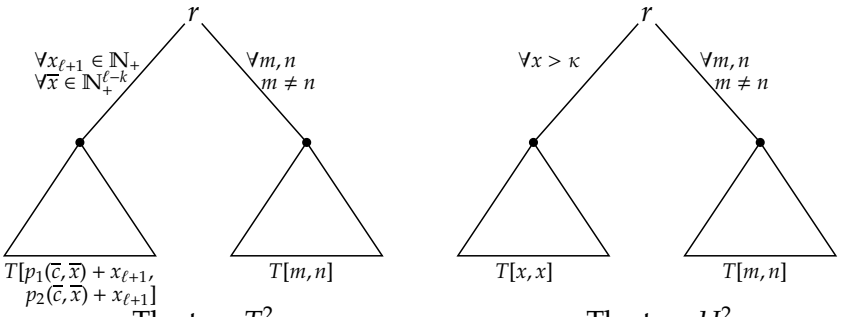
One might think that the first property suffices for proving Π^0_{2n-3}-hardness (with $i = n$) and that the trees U^i_m for $m < \omega$ are redundant. But we need these trees in order to carry out the inductive step. We also need the following property for the construction.

(P3) No leaf of any of the trees $T^i_{\overline{c}}$ or U^i_κ is a child of the root.

In the following section, we will describe the trees $T^i_{\overline{c}}$ and U^i_κ of height at most i and prove (P1) and (P2). Condition (P3) will be obvious from the construction. The subsequent section is then devoted to prove the effective automaticity of these trees.

5.2.1 Construction of trees

We start with a few definitions: A forest is a disjoint union of trees. Let H_1 and H_2 be two forests. The forest H_1^ω is the disjoint union of countably many copies of H_1. Formally, if $H_1 = (V, E)$, then $H_1^\omega = (V \times \mathbb{N}, E')$ with $((v, i), (w, j)) \in E'$ if and only if $(v, w) \in E$ and $i = j$. We write $H_1 \sim H_2$ for $H_1^\omega \cong H_2^\omega$. Then $H_1 \sim H_2$ if they are formed, up to isomorphism, by the same set of trees (i.e., any tree is isomorphic to some connected component of H_1 if and only if it is isomorphic to some connected component of H_2). If H is a forest and r does not belong to the domain of H, then we denote with $r \circ H$ the tree that results from adding r to H as new least element.



The tree $T^2_{\overline{c}}$ The tree U^2_κ

Figure 5.1: The tree $T^2_{\overline{c}}$ and U^2_κ

5.2.1.1 Induction base: construction of $T^2_{\overline{c}}$ and U^2_κ

For notational simplicity, we write k for $1 + 2(n-2)$. Hence, P_2 is a k-ary predicate. By Matiyasevich's theorem, we find two non-zero polynomials $p_1(x_1, \ldots, x_\ell)$, $p_2(x_1, \ldots, x_\ell) \in \mathbb{N}[\overline{x}]$, $\ell > k$, such that for any $\overline{c} \in \mathbb{N}_+^k$:

$$P_2(\overline{c}) \text{ holds} \iff \forall \overline{x} \in \mathbb{N}_+^{\ell-k} : p_1(\overline{c}, \overline{x}) \neq p_2(\overline{c}, \overline{x}) \,.$$

For two numbers $m, n \in \mathbb{N}_+$, let $T[m,n]$ denote the tree of height 1 with exactly $C(m,n)$ leaves, where C is the injective polynomial function from (5.1). Then define the following forests:

$$\begin{aligned} H^2 &= \biguplus\{T[m,n] \mid m,n \in \mathbb{N}_+, m \neq n\} \\ H^2_{\overline{c}} &= H^2 \uplus \biguplus\{T[p_1(\overline{c},\overline{x}) + x_{\ell+1}, p_2(\overline{c},\overline{x}) + x_{\ell+1}] \mid \overline{x} \in \mathbb{N}_+^{\ell-k}, x_{\ell+1} \in \mathbb{N}_+\} \\ J^2_\kappa &= H^2 \uplus \biguplus\{T[x,x] \mid x \in \mathbb{N}_+, x > \kappa\} \quad \text{for } \kappa \in \mathbb{N}_+ \cup \{\omega\} \end{aligned}$$

Note that $J^2_\omega = H^2$. Moreover, the forests J^2_κ ($\kappa \in \mathbb{N}_+ \cup \{\omega\}$) are pairwise non-isomorphic, since C is injective.

The trees $T^2_{\overline{c}}$ and U^2_κ, resp., are obtained from $H^2_{\overline{c}}$ and J^2_κ, resp., by multiplying all trees in these forests countably many times and adding a root afterwards:

$$T^2_{\overline{c}} = r \circ (H^2_{\overline{c}})^\omega \qquad U^2_\kappa = r \circ (J^2_\kappa)^\omega, \tag{5.5}$$

see Figure 5.1.

The following lemma (stating (P1) for the Π^0_1-predicate P_2 , i.e., for $i = 2$) can be proved in a similar way as Theorem 5.1.5.

Lemma 5.2.4 *For any $\overline{c} \in \mathbb{N}_+^k$, we have:*

$$P_2(\overline{c}) \text{ holds} \iff H^2_{\overline{c}} \sim J^2_\omega \iff T^2_{\overline{c}} \cong U^2_\omega \,.$$

Proof. By (5.5), it suffices to show the first equivalence. So first assume $P_2(\overline{c})$ holds. We have to prove that the forests $H^2_{\overline{c}}$ and $J^2_\omega = H^2$ contain the same trees (up to isomorphism). Clearly, every tree from H^2 is contained in $H^2_{\overline{c}}$. For the other direction, let $\overline{x} \in \mathbb{N}_+^{\ell-k}$ and $x_{\ell+1} \in \mathbb{N}_+$. Then the tree $T[p_1(\overline{c},\overline{x}) + x_{\ell+1}, p_2(\overline{c},\overline{x}) + x_{\ell+1}]$ occurs in $H^2_{\overline{c}}$. Since $P_2(\overline{c})$ holds, we have $p_1(\overline{c},\overline{x}) \neq p_2(\overline{c},\overline{x})$ and therefore $p_1(\overline{c},\overline{x}) + x_{\ell+1} \neq p_2(\overline{c},\overline{x}) + x_{\ell+1}$. Hence this tree also occurs in H^2.

Conversely suppose $H^2_{\overline{c}} \sim H^2$ and let $\overline{x} \in \mathbb{N}_+^{\ell-k}$. Then the tree $T[p_1(\overline{c},\overline{x}) + 1, p_2(\overline{c},\overline{x}) + 1]$ occurs in $H^2_{\overline{c}}$ and therefore in H^2. Hence $p_1(\overline{c},\overline{x}) \neq p_2(\overline{c},\overline{x})$. Since $\overline{x}$ was chosen arbitrarily, this implies $P_2(\overline{c})$. ∎

Now consider the forest $H^2_{\overline{c}}$ once more. If it contains a tree of the form $T[m,m]$ for some m (necessarily $m \geq 2$), then it contains all trees $T[x,x]$ for $x \geq m$. Hence, the forest $H^2_{\overline{c}}$ is isomorphic to one of the forests J^2_κ for some $\kappa \in \mathbb{N}_+ \cup \{\omega\}$. Hence with Lemma 5.2.4 we get:

$$P_2(\overline{c}) \text{ does not hold} \iff H^2_{\overline{c}} \not\sim J^2_\omega \iff \exists m \in \mathbb{N}_+ : H^2_{\overline{c}} \sim J^2_m$$

Hence we proved the following lemma, which states (P2) for the Π^0_1-predicate P_2, i.e., for $i = 2$.

Lemma 5.2.5 *For any $\overline{c} \in \mathbb{N}_+^k$, we have:*

$$P_2(\overline{c}) \textit{ does not hold} \iff \exists m \in \mathbb{N}_+ : T^2_{\overline{c}} \cong U^2_m \,.$$

This finishes the construction of the trees $T^2_{\overline{c}}$ and U^2_κ for $\kappa \in \mathbb{N}_+ \cup \{\omega\}$, and the verification of properties (P1) and (P2). Clearly, also (P3) holds for $T^2_{\overline{c}}$ and U^2_κ (all maximal paths have length 2).

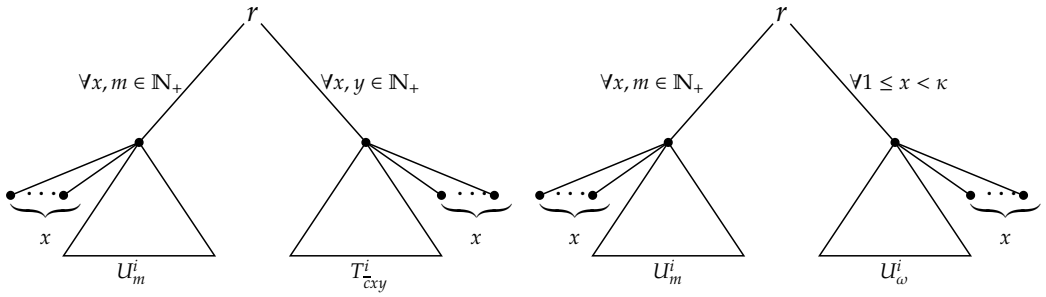
5.2.1.2 Induction step: construction of $T^{i+1}_{\overline{c}}$ and U^{i+1}_κ

For notational simplicity, we write again k for $1+2(n-i-1)$ such that P_{i+1} is a k-ary predicate and P_i a $(k+2)$-ary one.

We now apply the induction hypothesis. For any $\overline{c} \in \mathbb{N}_+^k$, $x, y \in \mathbb{N}_+$, $\kappa \in \mathbb{N}_+ \cup \{\omega\}$ let $T^i_{\overline{c}xy}$ and U^i_κ be trees of height at most i such that:

- $P_i(\overline{c}, x, y)$ holds if and only if $T^i_{\overline{c}xy} \cong U^i_\omega$.
- $P_i(\overline{c}, x, y)$ does not hold if and only if $T^i_{\overline{c}xy} \cong U^i_m$ for some $m \in \mathbb{N}_+$.

In a first step, we build the trees $T'_{\overline{c}xy}$ and $U'_{\kappa,x}$ ($x \in \mathbb{N}_+$) from $T^i_{\overline{c}xy}$ and U^i_κ, resp., by adding



The tree $T^{i+1}_{\overline{c}}$ The tree U^{i+1}_{κ}

Figure 5.2: The tree $T^{i+1}_{\overline{c}}$ and U^{i+1}_{κ}

x leaves as children of the root. This ensures

$$T'_{\overline{c}xy} \cong T'_{\overline{c}x'y'} \iff x = x' \wedge T^i_{\overline{c}xy} \cong T^i_{\overline{c}x'y'} \text{ and} \tag{5.6}$$

$$T'_{\overline{c}xy} \cong U'_{\kappa,x'} \iff x = x' \wedge T^i_{\overline{c}xy} \cong U^i_{\kappa}\,, \tag{5.7}$$

since, by property (P3), no leaf of any of the trees $T^i_{\overline{c}xy}$ or U^i_{κ} is a child of the root. Next, we collect these trees into forests as follows:

$$\begin{aligned} H^{i+1} &= \biguplus\{U'_{m,x} \mid x, m \in \mathbb{N}_+\}\,, \\ H^{i+1}_{\overline{c}} &= H^{i+1} \uplus \biguplus\{T'_{\overline{c}xy} \mid x, y \in \mathbb{N}_+\}\,, \text{ and} \\ J^{i+1}_{\kappa} &= H^{i+1} \uplus \biguplus\{U'_{\omega,x} \mid 1 \le x < \kappa\} \text{ for } \kappa \in \mathbb{N}_+ \cup \{\omega\}. \end{aligned}$$

The trees $T^{i+1}_{\overline{c}}$ and U^{i+1}_{κ}, resp., are then obtained from the forests $H^{i+1}_{\overline{c}}$ and J^{i+1}_{κ}, resp., by multiplying all trees in these forest countably many times and adding a root afterwards:

$$T^{i+1}_{\overline{c}} = r \circ (H^{i+1}_{\overline{c}})^{\omega} \quad \text{and} \quad U^{i+1}_{\kappa} = r \circ (J^{i+1}_{\kappa})^{\omega}, \tag{5.8}$$

see Figure 5.2.

Note that the height of any of these trees is one more than the height of the forests defining them and therefore at most $i+1$. Since none of the connected components of the forests $H^{i+1}_{\overline{c}}$ and J^{i+1}_{κ} is a singleton, none of the trees in (5.8) has a leaf that is a child of the root and therefore (P3) holds.

Lemma 5.2.6 *For all $\overline{c} \in \mathbb{N}_+^k$ we have*

$$P_{i+1}(\overline{c}) \text{ holds} \iff H^{i+1}_{\overline{c}} \sim J^{i+1}_{\omega} \iff T^{i+1}_{\overline{c}} \cong U^{i+1}_{\omega}\,.$$

Proof. Again, we only have to prove the first equivalence.

First assume $H^{i+1}_{\overline{c}} \sim J^{i+1}_{\omega}$ and let $x \ge 1$ be arbitrary. We have to exhibit some $y \ge 1$ such

that $P_i(\overline{c}, x, y)$ holds. Note that $U'_{\omega,x}$ belongs to J^{i+1}_ω and therefore to $H^{i+1}_{\overline{c}}$. Since $U'_{\omega,x} \not\cong U'_{m,x'}$ for any $m, x, x' \in \mathbb{N}_+$, this implies the existence of $x', y' \geq 1$ with $T'_{\overline{c}x'y'} \cong U'_{\omega,x}$. By (5.7), this is equivalent with $x = x'$ and $T^i_{\overline{c}xy'} \cong U^i_\omega$. Now the induction hypothesis implies that $P_i(\overline{c}, x, y')$ holds. Since $x \geq 1$ was chosen arbitrarily, we can deduce $P_{i+1}(\overline{c})$.

Conversely suppose $P_{i+1}(\overline{c})$. Let T belong to $H^{i+1}_{\overline{c}}$. By the induction hypothesis, it is one of the trees $U'_{\kappa,x}$ for some $x \in \mathbb{N}_+$, $\kappa \in \mathbb{N}_+ \cup \{\omega\}$. In any case, it also belongs to J^{i+1}_ω. Hence it remains to show that any tree of the form $U'_{\omega,x}$ belongs to $H^{i+1}_{\overline{c}}$. So let $x \in \mathbb{N}_+$. Then, by $P_{i+1}(\overline{c})$, there exists $y \in \mathbb{N}_+$ with $P_i(\overline{c}, x, y)$. By the induction hypothesis, we have $T^i_{\overline{c}xy} \cong U^i_\omega$ and therefore $T'_{\overline{c}xy} \cong U'_{\omega,x}$ (which belongs to $H^{i+1}_{\overline{c}}$ by the very definition). ∎

Lemma 5.2.7 *For all $\overline{c} \in \mathbb{N}_+^k$ there exists $\kappa \in \mathbb{N}_+ \cup \{\omega\}$ such that $T^{i+1}_{\overline{c}} \cong U^{i+1}_\kappa$.*

Proof. It suffices to prove that $H^{i+1}_{\overline{c}} \sim J^{i+1}_\kappa$ for some $\kappa \in \mathbb{N}_+ \cup \{\omega\}$. Choose κ as the smallest value in $\mathbb{N}_+ \cup \{\omega\}$ such that

$$\forall x \geq \kappa \, \forall y : \neg P_i(\overline{c}, x, y)$$

holds. By property (ii) from Lemma 5.2.3 for P_i, we get

$$\forall 1 \leq x < \kappa \, \exists y : P_i(\overline{c}, x, y).$$

By the induction hypothesis, we get

$$\forall x \geq \kappa \, \forall y : T'_{\overline{c}xy} \not\cong U'_{\omega,x} \text{ and } \forall 1 \leq x < \kappa \, \exists y : T'_{\overline{c}xy} \cong U'_{\omega,x}\,.$$

It follows that $H^{i+1}_{\overline{c}}$ contains, apart from the trees in $H^{i+1} = \biguplus\{U'_{m,x} \mid x, m \in \mathbb{N}_+\}$, exactly the trees from $\biguplus\{U'_{\omega,x} \mid 1 \leq x < \kappa\}$. Hence, $H^{i+1}_{\overline{c}} \sim J^{i+1}_\kappa$. ∎

Lemma 5.2.6 and 5.2.7 immediately imply:

Lemma 5.2.8 *For all $\overline{c} \in \mathbb{N}_+^k$ we have*

$$P_{i+1}(\overline{c}) \text{ does not hold} \iff \exists m \in \mathbb{N}_+ : T^{i+1}_{\overline{c}} \cong U^{i+1}_m.$$

In summary, we obtained the following:

Proposition 5.2.9 *Let $n \geq 2$ and let $P(x)$ be a Π^0_{2n-3}-predicate. Then, for any $c \in \mathbb{N}_+$, we have*

$$P(c) \text{ holds} \iff T^n_c \cong U^n_\omega\,.$$

To infer the Π^0_{2n-3}-hardness of the isomorphism problem for $\mathcal{T}_n$ from this proposition, it remains to be shown that the trees T^n_c and U^n_ω are effectively automatic – this is the topic of the next section.

5.2.2 Automaticity

For constructing automatic presentations for the trees from the previous section, it is actually easier to work with *dags* (*directed acyclic graphs*). The *height* of a dag D is the length (number of edges) of a longest directed path in D. We only consider dags of finite height. A *root* of a dag is a node without incoming edges. A dag $D = (V, E)$ can be unfolded into a forest unfold(D) in the usual way: Nodes of unfold(D) are directed paths in D that cannot be extended to the left (i.e., the initial node of the path is a root) and there is an edge between a path p and a path p' if and only if p' extends p by one more node. For a node $v \in V$ of D, we define the tree unfold(D, v) as follows: First we restrict D to those nodes that are reachable from v and then we unfold the resulting dag. We need the following lemma.

Lemma 5.2.10 *From given $k \in \mathbb{N}$ and an automatic dag $D = (V, E)$ of height at most k, one can construct effectively an automatic presentation $\mathcal{P}$ with $\mathcal{S}(\mathcal{P}) \cong$ unfold(D).*

Proof. The universe for our automatic copy of unfold(D) is the set P of all convolutions $v_1 \otimes v_2 \otimes \cdots \otimes v_m$, where v_1 is a root and $(v_i, v_{i+1}) \in E$ for all $1 \leq i < m$. Since D has height at most k, we have $m \leq k$. Since the edge relation of D is automatic and since the set of all roots in D is first-order definable and hence regular, P is indeed a regular set. Moreover, the edge relation of unfold(D) becomes clearly FA recognizable on P. ∎

For $2 \leq i \leq n$, let us consider the following forest:

$$F^i = \biguplus \{T^i_{\overline{c}} \mid \overline{c} \in \mathbb{N}_+^{1+2(n-i)}\} \uplus \biguplus \{U^i_m \mid m \in \mathbb{N}_+ \cup \{\omega\}\} .$$

Technically, this section proves by induction over i the following statement:

Proposition 5.2.11 *From $\ell \in \mathbb{N}_+$, $p_1, p_2 \in \mathbb{N}[x_1, \ldots, x_\ell]$ and $2 \leq i \leq n$, we can compute an automatic copy $\mathcal{F}^i$ of F^i such that there exists an isomorphism $f^i : F^i \to \mathcal{F}^i$ that maps*

1. *the root of the tree $T^i_{\overline{c}}$ to $a^{\overline{c}}$ (for all $\overline{c} \in \mathbb{N}_+^{1+2(n-i)}$),*
2. *the root of the tree U^i_ω to ε, and*
3. *the root of the tree U^i_m to b^m (for all $m \in \mathbb{N}_+$).*

This will give the desired result since T^n_c is then isomorphic to the connected component of $\mathcal{F}^n$ that contains the word a^c (and similarly for U^n_κ). Note that this connected component is again automatic by Theorem 2.5.11, since the forest $\mathcal{F}^n$ has bounded height.

By Lemma 5.2.10, it suffices to construct an automatic dag $\mathcal{D}^i$ such that there is an isomorphism $h :$ unfold$(\mathcal{D}^i) \to \mathcal{F}^i$ that is the identity on the set of roots of $\mathcal{D}^i$.

5.2.2.1 Induction base: the automatic dag $\mathcal{D}^2$

Recall the definitions of Σ^a_ℓ, $a^{\overline{e}}$, and $\otimes_k(L)$ from Section 5.1.

Lemma 5.2.12 *From $\ell \in \mathbb{N}_+$, $q_1, q_2 \in \mathbb{N}[x_1, \ldots, x_\ell]$, and a symbol a, one can compute an automatic forest of height 1 over an alphabet $\Sigma^a_\ell \uplus \Gamma$ such that*

- *the roots are the words from $\otimes_\ell(a^+)$,*
- *the leaves are words from Γ^+, and*
- *the tree rooted at $a^{\overline{e}}$ is isomorphic to $T[q_1(\overline{e}), q_2(\overline{e})]$.*

Proof. Set $p(x_1, \ldots, x_\ell) = C(q_1(x_1, \ldots, x_\ell), q_2(x_1, \ldots, x_\ell))$ and recall the definition of the automata $\mathcal{A}[p]$ and $\mathrm{Run}_{\mathcal{A}[p]}$ from Section 5.1. Recall also that we let π be the projection with $\pi(p, a, q) = a$ for a transition (p, a, q) of $\mathcal{A}[p]$. Then let

$$\begin{aligned} L[q_1, q_2] &= \otimes_\ell(a^+) \cup (\pi^{-1}(\otimes_\ell(a^+)) \cap L(\mathrm{Run}_{\mathcal{A}[p]})) \text{ and} \\ E[q_1, q_2] &= \{(u, v) \mid u \in \otimes_\ell(a^+), v \in \pi^{-1}(u) \cap L(\mathrm{Run}_{\mathcal{A}[p]})\} . \end{aligned}$$

Then $L[q_1, q_2]$ is regular and $E[q_1, q_2]$ is automatic, i.e., the pair $(L[q_1, q_2]; E[q_1, q_2])$ is an automatic graph. It is actually a forest of height 1, the words from $\otimes_\ell(a^+)$ form the roots, and the tree rooted at $a^{\overline{e}}$ has precisely $p(\overline{e})$ leaves, i.e., it is isomorphic to $T[q_1(\overline{e}), q_2(\overline{e})]$. ∎

From now on, we use the notations from Section 5.2.1.1. Using Lemma 5.2.12, we can compute automatic forests $\mathcal{F}_1$ and $\mathcal{F}_2$ over alphabets $\Sigma^a_{\ell+1} \uplus \Gamma_1$ and $\Sigma^b_2 \uplus \Gamma_2$, respectively, such that

(a) the roots of $\mathcal{F}_1$ are the words from $\otimes_{\ell+1}(a^+)$,

(b) the roots of $\mathcal{F}_2$ are the words from $\otimes_2(b^+)$,

(c) the leaves of $\mathcal{F}_i$ are words from Γ_i^+ $(i \in \{1, 2\})$,

(d) the tree rooted at $a^{\overline{e}e_{\ell+1}}$ is isomorphic to $T[p_1(\overline{e}) + e_{\ell+1}, p_2(\overline{e}) + e_{\ell+1}]$ for $\overline{e} \in \mathbb{N}_+^\ell$, $e_{\ell+1} \in \mathbb{N}_+$,

(e) the tree rooted at $b^{e_1 e_2}$ is isomorphic to $T[e_1, e_2]$ for $e_1, e_2 \in \mathbb{N}_+$.

We can assume that the alphabets Γ_1, Γ_2, $\Sigma^a_{\ell+1}$, and Σ^b_2 are mutually disjoint. Let $\mathcal{F} = (V_\mathcal{F}, E_\mathcal{F})$ be the disjoint union of $\mathcal{F}_1$ and $\mathcal{F}_2$; it is effectively automatic.

The universe of the automatic dag $\mathcal{D}^2$ is the regular language

$$\otimes_k(a^+) \cup b^* \cup (\$^* \otimes V_\mathcal{F}),$$

where \$ is a new symbol. We have the following edges:

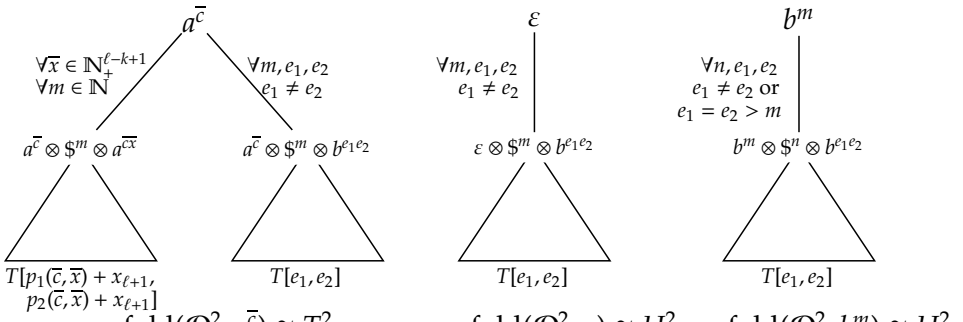


Figure 5.3: Automatic presentation of $T^2_{\overline{c}}$ and U^2_κ

- For $u, v \in V_{\mathcal{F}}$, $\$^m \otimes u$ is connected to $\$^n \otimes v$ if and only if $m = n$ and $(u, v) \in E_{\mathcal{F}}$. This produces α_0 many copies of $\mathcal{F}$.
- $a^{\overline{c}}$ is connected to any word from $\$^* \otimes (\{a^{\overline{c}\,\overline{x}} \mid \overline{x} \in \mathbb{N}_+^{\ell-k+1}\} \cup \{b^{e_1 e_2} \mid e_1 \neq e_2\})$. By point (d) and (e) above, this means that the tree $\mathrm{unfold}(\mathcal{D}^2, a^{\overline{c}})$ has $\aleph_0$ many subtrees isomorphic to $T[p_1(\overline{c}\,\overline{x}) + x_{\ell+1}, p_2(\overline{c}\,\overline{x}) + x_{\ell+1}]$ for $\overline{x} \in \mathbb{N}_+^{\ell-k}$, $x_{\ell+1} \in \mathbb{N}_+$ and $T[e_1, e_2]$ for $e_1, e_2 \in \mathbb{N}_+$, $e_1 \neq e_2$. Hence, $\mathrm{unfold}(\mathcal{D}^2, a^{\overline{c}}) \cong T^2_{\overline{c}}$.
- ε is connected to all words from $\$^* \otimes \{b^{e_1 e_2} \mid e_1 \neq e_2\}$. By (e) above, this means that the tree $\mathrm{unfold}(\mathcal{D}^2, \varepsilon)$ has $\aleph_0$ many subtrees isomorphic to $T[e_1, e_2]$ for $e_1, e_2 \in \mathbb{N}_+$, $e_1 \neq e_2$. Hence, $\mathrm{unfold}(\mathcal{D}^2, \varepsilon) \cong U^2_\omega$.
- b^m ($m \in \mathbb{N}_+$) is connected to all words from $\$^* \otimes \{b^{e_1 e_2} \mid e_1 \neq e_2 \text{ or } e_1 = e_2 > m\}$. By (e) above, this means that the tree $\mathrm{unfold}(\mathcal{D}^2, b^m)$ has $\aleph_0$ many subtrees isomorphic to $T[e_1, e_2]$ for all $e_1, e_2 \in \mathbb{N}_+$ with $e_1 \neq e_2$ or $e_1 = e_2 > m$. Hence, $\mathrm{unfold}(\mathcal{D}^2, b^m) \cong U^2_m$.

Thus, $\mathrm{unfold}(\mathcal{D}_2) \cong F^2$ and the roots are as required in Proposition 5.2.11, see Figure 5.3. Moreover, it is clear that $\mathcal{D}_2$ is automatic.

5.2.2.2 Induction step: the automatic dag $\mathcal{D}^{i+1}$

Suppose $\mathcal{D}^i = (V, E)$ is such that $\mathcal{F}^i = \mathrm{unfold}(\mathcal{D}^i)$ is as described in Proposition 5.2.11.

We use the notations from Section 5.2.1.2. We first build another automatic dag $\mathcal{D}'$, whose unfolding will comprise (copies of) all the trees $U'_{\kappa,x}$ ($\kappa \in \mathbb{N}_+ \cup \{\omega\}$, $x \in \mathbb{N}_+$) and $T'_{\overline{c}xy}$ ($\overline{c} \in \mathbb{N}_+^k$, $x, y \in \mathbb{N}_+$). Recall that the set of roots of $\mathcal{D}^i$ is $\otimes_{k+2}(a^+) \cup b^* \subseteq V$. The universe of $\mathcal{D}'$ consists of the regular language

$$(V \setminus b^*) \cup (\sharp^+ \otimes b^*) \cup \sharp_1^+ \sharp_2^*,$$

where $\sharp, \sharp_1$, and $\sharp_2$ are new symbols. We have the following edges in $\mathcal{D}'$:

– All edges from E except those with an initial node in b^* are present in $\mathcal{D}'$.

– $a^{\overline{c}xy} \in V$ is connected to all words of the form $\sharp_1^i \sharp_2^{x-i}$ for $\overline{c} \in \mathbb{N}_+^k$, $x, y \in \mathbb{N}_+$, and $1 \leq i \leq x$. This ensures that the subtree rooted at $a^{\overline{c}xy}$ gets x new leaves, which are children of the root. Hence $\mathrm{unfold}(\mathcal{D}', a^{\overline{c}xy}) \cong T'_{\overline{c}xy}$.

– $\sharp^x \otimes b^m$ for $x \in \mathbb{N}_+$ and $m \in \mathbb{N}$ is connected to (i) all nodes to which b^m is connected in $\mathcal{D}^i$ and to (ii) all nodes from $\sharp_1^i \sharp_2^{x-i}$ for $1 \leq i \leq x$. This ensures that $\mathrm{unfold}(\mathcal{D}', \sharp^x \otimes b^m) \cong U'_{m,x}$ in case $m \in \mathbb{N}_+$ and $\mathrm{unfold}(\mathcal{D}', \sharp^x \otimes \varepsilon) \cong U'_{\omega,x}$.

In summary, $\mathcal{D}'$ is an dag, whose unfolding consists of (a copy of) $U'_{\omega,x}$ rooted at $\sharp^x \otimes \varepsilon$, $U'_{m,x}$ ($m \in \mathbb{N}_+$) rooted at $\sharp^x \otimes b^m$, and $T'_{\overline{c}xy}$ rooted at $a^{\overline{c}xy}$.

From the automatic dag $\mathcal{D}'$, we now build in a final step the automatic dag $\mathcal{D}^{i+1}$. This is very similar to the constructions of $\mathcal{D}^2$ and $\mathcal{D}'$ above. Let V' be the universe of $\mathcal{D}'$. The universe of $\mathcal{D}^{i+1}$ is the regular language

$$\otimes_k(a^+) \cup b^* \cup (\$^* \otimes V') \,.$$

The edges are as follows:

– For $u, v \in V'$, $\$^m \otimes u$ is connected to $\$^n \otimes v$ if and only if $m = n$ and (u, v) is an edge of $\mathcal{D}'$. This generates $\aleph_0$ many copies of $\mathcal{D}'$.

– $a^{\overline{c}}$ is connected to every word from $\$^* \otimes (\{a^{\overline{c}xy} \mid x, y \in \mathbb{N}_+\} \cup (\sharp^+ \otimes b^+))$. Hence, the tree $\mathrm{unfold}(\mathcal{D}^{i+1}, a^{\overline{c}})$ has $\aleph_0$ many subtrees isomorphic to $T'_{\overline{c}xy}$ for $x, y \in \mathbb{N}_+$ and $U'_{m,x}$ for $x, m \in \mathbb{N}_+$. Thus, $\mathrm{unfold}(\mathcal{D}^{i+1}, a^{\overline{c}}) \cong T^{i+1}_{\overline{c}}$.

– ε is connected to all words from $\$^* \otimes (\sharp^+ \otimes b^*)$. Hence, the tree $\mathrm{unfold}(\mathcal{D}^{i+1}, \varepsilon)$ has $\aleph_0$ many subtrees isomorphic to $U'_{\kappa,x}$ for all $x \in \mathbb{N}_+$ and $\kappa \in \mathbb{N}_+ \cup \{\omega\}$. Thus, $\mathrm{unfold}(\mathcal{D}^{i+1}, \varepsilon) \cong U^{i+1}_{\omega}$.

– b^m ($m \in \mathbb{N}_+$) is connected to all words from $\$^* \otimes ((\sharp^+ \otimes b^+) \cup \{\sharp^x \otimes \varepsilon \mid 1 \leq x < m\})$. This means that the tree $\mathrm{unfold}(\mathcal{D}^{i+1}, b^m)$ has $\aleph_0$ many subtrees isomorphic to $U'_{m,x}$ for all $m, x \in \mathbb{N}_+$ and $U'_{\omega,x}$ for all $1 \leq x < m$. Hence, $\mathrm{unfold}(\mathcal{D}^{i+1}, b^m) \cong U^{i+1}_m$.

See Figure 5.4, 5.5, and 5.6 for the overall construction. This finishes the proof of Proposition 5.2.11. Hence we obtain:

Theorem 5.2.13 *1. For any $n \geq 2$, the isomorphism problem for automatic trees of height at most n is Π^0_{2n-3}-complete.*

2. The isomorphism problem for the class of automatic trees of finite height is computably equivalent to $\mathsf{FOTh}(\mathbb{N}; +, \times)$.

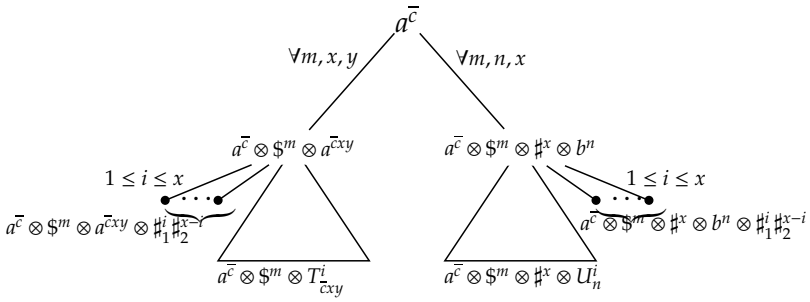


Figure 5.4: Automatic presentation of $T^{i+1}_{\overline{c}}$

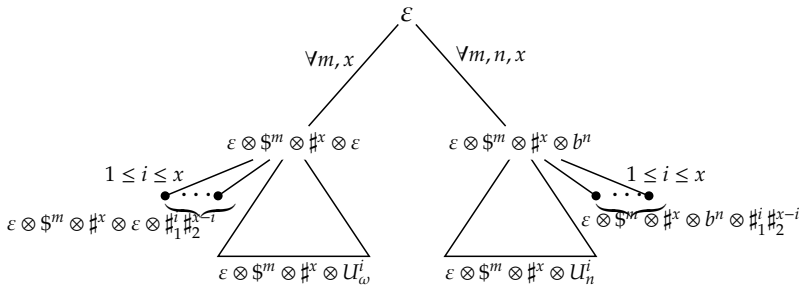


Figure 5.5: Automatic presentation of U^{i+1}_ω

Proof. We first prove the first statement. Containment in Π^0_{2n-3} was shown in Proposition 5.2.2. For the hardness, let $P \subseteq \mathbb{N}_+$ be any Π^0_{2n-3}-predicate and let $c \in \mathbb{N}_+$. Then, above, we constructed the automatic forest $\mathcal{F}^n$ of height n. The trees T^n_c and U^n_ω are first-order definable in $\mathcal{F}^n$ since they are (isomorphic to) the trees rooted at $a^{\overline{c}}$ and ε, resp. Hence these two trees are automatic. By Proposition 5.2.9, they are isomorphic if and only if $P(c)$ holds.

We now come to the second statement. Since the proof of the first statement is uniform in the level n, we can compute from two automatic trees T_1, T_2 of finite height an arithmetical formula, which is true if and only if $T_1 \cong T_2$. For the other direction, one observes that the height of an automatic tree of finite height can be computed. Then the result follows from the first statement because of the uniformity of its proof. ∎

5.3 Computable trees of finite height

In this section, we briefly discuss the isomorphism problem for computable trees of finite height.

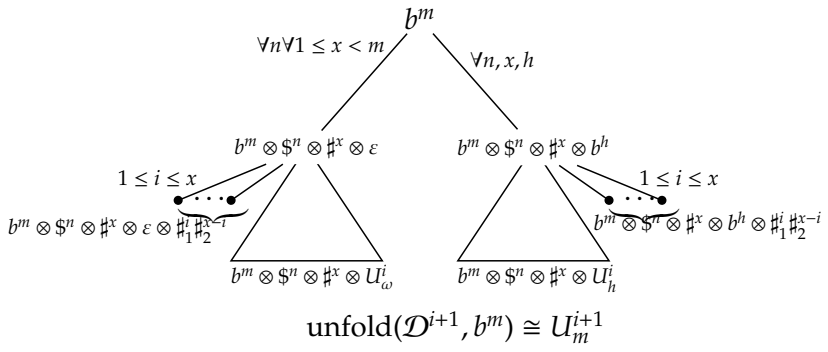


Figure 5.6: Automatic presentation of U_m^{i+1}

Theorem 5.3.1 *For every $n \geq 1$, the isomorphism problem for computable trees of height at most n is Π^0_{2n}-complete.*

Proof. For the upper bound, let us first assume that $n = 1$. Two computable trees T_1 and T_2 of height 1 are isomorphic if and only if: for every $k \geq 0$, there exist at least k nodes in T_1 if and only if there exist at least k nodes in T_2. This is a Π^0_2-statement. For the inductive step, we can reuse the arguments from the proof of Proposition 5.2.2.

For the lower bound, we first note that the isomorphism problem for computable trees of height 1 is Π^0_2-complete. It is known that the problem whether a given computably enumerable set is infinite is Π^0_2-complete (See Example 2.3.4). For a given deterministic Turing-machine M, we construct a computable tree $T(M)$ of height 1 as follows: the set of leaves of $T(M)$ is the set of all accepting computations of M. We add a root to the tree and connect the root to all leaves. If $L(M)$ is infinite, then $T(M)$ is isomorphic to the height-1 tree with infinitely many leaves. If $L(M)$ is finite, then there exists $m \in \mathbb{N}$ such that $T(M)$ is isomorphic to the height-1 tree with m leaves. We can use this construction as the base case for our construction in Section 5.2.1.2. This yields the lower bound for all $n \geq 1$. ∎

5.4 Automatic linear orders

Let $I = (D_I; \leq_I)$ be a linear order and let $\mathcal{L} = \{L_i \mid i \in D_I\}$ be a class of linear orders, where $L_i = (D_i; \leq_i)$ for $i \in D_I$. The *sum* $\sum \mathcal{L}$ is the linear order $(\{(x, i) \mid i \in D_I, x \in D_i\}; \leq)$ where for all $i, j \in D_I$, $x \in D_i$, and $y \in D_j$,

$$(x, i) \leq (y, j) \iff i <_I j \vee (i = j \wedge x \leq_i y) .$$

We use $L_1 + L_2$ to denote $\sum\{L_i \mid i \in \mathbf{2}\}$. We denote with $L_1 \cdot L_2$ the sum $\sum\{L_1^i \mid i \in L_2\}$ where $L_1^i \cong L_1$ for every $i \in L_2$. An *interval* of a linear order $L = (D; \leq)$ is a subset $I \subseteq D$ such that $x, y \in I$ and $x < z < y$ imply $z \in I$.

Recall from Example 2.4.2 that $\leq_{\text{lex}}$ denotes the lexicographic order on words. For convenience, we use $\leq_{\text{lex}}$ regardless of the corresponding alphabets and orders on the alphabets. The precise definitions of $\leq_{\text{lex}}$ in different occurrences will be clear from the context.

This section is devoted to proving that the isomorphism problem on the class of automatic linear orders is at least as hard as $\mathsf{FOTh}(\mathbb{N}; +, \times)$. To this end, it suffices to prove (uniformly in n) Σ^0_n-hardness for every even n. The general plan for this is similar to the proof for trees of finite height: we use Hilbert's 10^{th} problem to handle Π^0_1-predicates in several variables and an inductive construction of more complicated linear orders to handle quantifiers, i.e., to proceed from a Π^0_{2i-1}- to a Σ^0_{2i}-predicate (and from a Σ^0_{2i}- to a Π^0_{2i+1}-predicate).

So let $n \geq 1$ be even and let $P_n(x_0)$ be a Σ^0_n-predicate. For every odd (even) number $1 \leq i < n$, let $P_i(x_0, \ldots, x_{n-i})$ be the Π^0_i-predicate (Σ^0_i-predicate) such that $P_{i+1}(x_0, \ldots, x_{n-i-1})$ is logically equivalent to $Qx_{n-i} : P_i(x_0, \ldots, x_{n-i})$ where $Q = \exists$ if i is odd and $Q = \forall$ if i is even. We fix these predicates for the rest of Section 5.4.

By induction on $1 \leq i \leq n$, we will construct from $\overline{c} \in \mathbb{N}_+^{n-i+1}$ the following linear orders:

- a test linear order $L^i_{\overline{c}}$,
- a linear order K^i, and
- a set of linear orders $\mathcal{M}^i$ such that $\mathcal{M}^1 = \{M^1_m \mid m \in \mathbb{N}_+\}$ and $\mathcal{M}^i$ is the singleton $\{M^i\}$ if $i > 1$.

These linear orders will have the following properties:

(P1) $P_i(\overline{c})$ holds if and only if $L^i_{\overline{c}} \cong K^i$.

(P2) $P_i(\overline{c})$ does not hold if and only if $L^i_{\overline{c}} \cong M$ for some $M \in \mathcal{M}^i$.

(P3) The linear order $\omega \cdot \mathbf{i}$ is not isomorphic to any interval of $L^i_{\overline{c}}, K^i, M$ where $M \in \mathcal{M}^i$.

In the rest of the section, we will inductively construct $L^i_{\overline{c}}$, K^i, and $\mathcal{M}^i$ and prove (P1), (P2), and (P3). The subsequent section is devoted to proving the effective automaticity of these linear orders.

5.4.1 Construction of linear orders

Our construction of linear orders is quite similar to the construction for trees from Section 5.2.1. One of the main differences is that in the inductive step for trees, we went from a Π^0_i-predicate directly to a Π^0_{i+2}-predicate. Thereby the height of the trees only increased by one. This was crucial in order to get Π^0_{2n-3}-completeness for the isomorphism problem for automatic trees of height $n \geq 2$. For automatic linear orders, we split the construction into

two inductive steps: in the first step, we go from a Π^0_i-predicate (i odd) to a Σ^0_{i+1}-predicate, whereas in the second step, we go from a Σ^0_{i+1}-predicate to a Π^0_{i+2}-predicate.

A key technique used in the construction is the shuffle sum of a class of linear orders. Let I be a countable set. A *dense I-coloring* of $\mathbb{Q}$ is a mapping $c : \mathbb{Q} \to I$ such that for all $x, y \in \mathbb{Q}$ with $x < y$ and all $i \in I$ there exists $x < z < y$ with $c(z) = i$.

Definition 5.4.1 *Let $\mathcal{L} = \{L_i \mid i \in I\}$ be a set of linear orders with I countable and let $c : \mathbb{Q} \to I$ be a dense I-coloring of $\mathbb{Q}$. The* shuffle sum *of $\mathcal{L}$, denoted* $\mathrm{Shuf}(\mathcal{L})$*, is the linear order $\sum_{x\in\mathbb{Q}} L_{c(x)}$.*

In the above definition, the isomorphism type of $\sum_{x\in\mathbb{Q}} L_{c(x)}$ does not depend on the choice of the dense I-coloring c, see e.g. [101]. Hence $\mathrm{Shuf}(\mathcal{L})$ is indeed uniquely defined.

In this section, we will consider classes $\mathcal{L}_1$ and $\mathcal{L}_2$ of linear orders that we consider as classes of isomorphism types. Therefore, we use the following abbreviations:

- "$L \in \mathcal{L}_1$" denotes that $\mathcal{L}_1$ contains a linear order isomorphic to L,
- "$\mathcal{L}_1 \subseteq \mathcal{L}_2$" denotes $\forall L_1 \in \mathcal{L}_1\ \exists L_2 \in \mathcal{L}_2 : L_1 \cong L_2$, and
- "$\mathcal{L}_1 = \mathcal{L}_2$" abbreviates $\mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_1$.

5.4.1.1 Induction base: construction of $L^1_{\overline{c}}$, K^1, and M^1_m

Recall from Section 5.1 that the polynomial function $C(x, y) = (x + y)^2 + 3x + y$ is injective. For $n_1, n_2 \in \mathbb{N}_+$, let $L[n_1, n_2]$ be the finite linear order of length $C(n_1, n_2)$.

By applying Matiyasevich's theorem, we obtain two polynomials $p_1(\overline{x}), p_2(\overline{x}) \in \mathbb{N}[\overline{x}]$ in ℓ variables, $\ell > n$, such that for all $\overline{c} \in \mathbb{N}^n_+$, the Π^0_1-predicate $P_1(\overline{c})$ holds if and only if

$$\forall \overline{x} \in \mathbb{N}^{\ell-n} :\ p_1(\overline{c}, \overline{x}) \neq p_2(\overline{c}, \overline{x})\ .$$

Fix $\overline{c} \in \mathbb{N}^n_+$ and $m \in \mathbb{N}_+$. We define the following four classes of finite linear orders:

$$\mathcal{L}^1_1(\overline{c}) = \{L[p_1(\overline{c}, \overline{x}) + x_{\ell+1}, p_2(\overline{c}, \overline{x}) + x_{\ell+1}] \mid \overline{x} \in \mathbb{N}^{\ell-n}_+, x_{\ell+1} \in \mathbb{N}_+\} \tag{5.9}$$

$$\mathcal{L}^1_2(m) = \{L[x + m, x + m] \mid x \in \mathbb{N}_+\} \tag{5.10}$$

$$\mathcal{L}^1_3 = \{L[x + y, x] \mid x, y \in \mathbb{N}_+\} \tag{5.11}$$

$$\mathcal{L}^1_4 = \{L[x, x + y] \mid x, y \in \mathbb{N}_+\} \tag{5.12}$$

The linear orders $L^1_{\overline{c}}$, K^1, and M^1_m are obtained by taking the shuffle sums of unions of the above classes of linear orders:

$$L^1_{\overline{c}} = \mathrm{Shuf}(\mathcal{L}^1_1(\overline{c}) \cup \mathcal{L}^1_3 \cup \mathcal{L}^1_4), \qquad K^1 = \mathrm{Shuf}(\mathcal{L}^1_3 \cup \mathcal{L}^1_4), \qquad M^1_m = \mathrm{Shuf}(\mathcal{L}^1_2(m) \cup \mathcal{L}^1_3 \cup \mathcal{L}^1_4).$$

The next lemma is needed to prove (P1) and (P2) for the Π^0_1-predicate P_1.

Lemma 5.4.2 *Suppose $\mathcal{L}_1$ and $\mathcal{L}_2$ are two countable sets of finite linear orders. Then*

$$\mathcal{L}_1 = \mathcal{L}_2 \iff \mathrm{Shuf}(\mathcal{L}_1) \cong \mathrm{Shuf}(\mathcal{L}_2)$$

and no interval of $\mathrm{Shuf}(\mathcal{L}_1)$ *is isomorphic to* ω.

Proof. If $\mathcal{L}_1 = \mathcal{L}_2$, then it is clear that $\mathrm{Shuf}(\mathcal{L}_1) \cong \mathrm{Shuf}(\mathcal{L}_2)$. Conversely, suppose there exists an isomorphism f from $\mathrm{Shuf}(\mathcal{L}_1)$ to $\mathrm{Shuf}(\mathcal{L}_2)$. We prove below that $\mathcal{L}_1 = \mathcal{L}_2$. By symmetry we only need to prove $\mathcal{L}_1 \subseteq \mathcal{L}_2$.

Note that for $i \in \{1, 2\}$, $\mathrm{Shuf}(\mathcal{L}_i)$ is obtained by replacing each $q \in \mathbb{Q}$ with some linear order L_q^i (whose type is) contained in $\mathcal{L}_i$. For every $q \in \mathbb{Q}$, if $f(L_q^1)$ contains elements from L_p^2 and $L_{p'}^2$ for some $p < p'$, then $f(L_q^1)$ is infinite which is impossible. Therefore f maps L_q^1 into L_p^2 for some $p \in \mathbb{Q}$. Using the same argument with f replaced by f^{-1}, we can also prove that f^{-1} maps L_p^2 into L_q^1. Hence $L_q^1 \cong L_p^2$. This means that for all $L \in \mathcal{L}_1$, there is $L' \in \mathcal{L}_2$ such that $L \cong L'$. Therefore $\mathcal{L}_1 \subseteq \mathcal{L}_2$.

If $x_1 < x_2 < \cdots$ in $\mathrm{Shuf}(\mathcal{L}_1)$, then there are $p < p'$ in $\mathbb{Q}$ and $k < \ell$ in $\mathbb{N}_+$ such that $x_k \in L_p^1$ and $x_\ell \in L_{p'}^1$. But then the interval $[x_k, x_\ell]$ is infinite. Hence no interval in $\mathrm{Shuf}(\mathcal{L}_1)$ is isomorphic to ω. ∎

The next lemma states (P1) and (P2) for $i = 1$:

Lemma 5.4.3 *For any $\overline{c} \in \mathbb{N}_+^n$, we have:*

(1) $P_1(\overline{c})$ holds $\iff L_{\overline{c}}^1 \cong K^1$.

(2) $P_1(\overline{c})$ does not hold $\iff \exists m \in \mathbb{N}_+ : L_{\overline{c}}^1 \cong M_m^1$.

Proof. For (1), we have

$$\begin{aligned}
P_1(\overline{c}) &\iff \forall \overline{x} \in \mathbb{N}_+^{\ell-n} : p_1(\overline{c}, \overline{x}) \neq p_2(\overline{c}, \overline{x}) \\
&\iff \forall \overline{x} \in \mathbb{N}_+^{\ell-n}, x_{\ell+1} \in \mathbb{N}_+ : p_1(\overline{c}, \overline{x}) + x_{\ell+1} \neq p_2(\overline{c}, \overline{x}) + x_{\ell+1} \\
&\iff \forall \overline{x} \in \mathbb{N}_+^{\ell-n}, x_{\ell+1} \in \mathbb{N}_+ : L[p_1(\overline{c}, \overline{x}) + x_{\ell+1}, p_2(\overline{c}, \overline{x}) + x_{\ell+1}] \in \mathcal{L}_3^1 \cup \mathcal{L}_4^1 \\
&\iff \mathcal{L}_1^1(\overline{c}) \cup \mathcal{L}_3^1 \cup \mathcal{L}_4^1 = \mathcal{L}_3^1 \cup \mathcal{L}_4^1 \\
&\overset{\text{Lemma 5.4.2}}{\iff} \mathcal{L}_{\overline{c}}^1 \cong K^1.
\end{aligned}$$

For (2), we get

$$\begin{aligned}
\neg P_1(\overline{c}) &\iff \exists \overline{x} \in \mathbb{N}_+^{\ell-n} : p_1(\overline{c},\overline{x}) = p_2(\overline{c},\overline{x}) \\
&\iff \exists m \in \mathbb{N}_+ : L[m+1, m+1] \in \mathcal{L}_1^1(\overline{c}) \\
&\iff \exists m \in \mathbb{N}_+ : (\forall k > m : L[k,k] \in \mathcal{L}_1^1(\overline{c}) \wedge \forall 1 \leq k \leq m : L[k,k] \notin \mathcal{L}_1^1(\overline{c})) \\
&\iff \exists m \in \mathbb{N}_+ : \mathcal{L}_1^1(\overline{c}) \cup \mathcal{L}_3^1 \cup \mathcal{L}_4^1 = \mathcal{L}_2^1(m) \cup \mathcal{L}_3^1 \cup \mathcal{L}_4^1 \\
&\overset{\text{Lemma 5.4.2}}{\iff} \exists m \in \mathbb{N}_+ : L_{\overline{c}}^1 \cong M_m^1 .
\end{aligned}$$

∎

Since $L_{\overline{c}}^1$, K^1, and M_m^1 are shuffle sums, they satisfy (P3) by Lemma 5.4.2. This finishes the construction for the base case.

5.4.1.2 First induction step: from P_i to P_{i+1} for i odd

Suppose $i \geq 1$ is an odd number. For notational simplicity, we write k for $n - i$. Thus, P_{i+1} is a k-ary predicate and P_i is a $(k+1)$-ary one. For all $\overline{c} \in \mathbb{N}_+^k$, $P_{i+1}(\overline{c})$ is logically equivalent to $\exists x : P_i(\overline{c}, x)$. Applying the inductive hypothesis, for any $\overline{c} \in \mathbb{N}_+^k$ and $x \in \mathbb{N}_+$, we obtain linear orders $L_{\overline{c}x}^i$, K^i, and the set $\mathcal{M}^i$ such that

- $P_i(\overline{c}, x)$ holds if and only if $L_{\overline{c}x}^i \cong K^i$,
- $P_i(\overline{c}, x)$ does not hold if and only if $L_{\overline{c}x}^i \cong M$ for some $M \in \mathcal{M}^i$, and
- $\omega \cdot \mathbf{i}$ is not isomorphic to any interval of $L_{\overline{c}x}^i$, K^i, or M where $M \in \mathcal{M}^i$.

Fix $\overline{c} \in \mathbb{N}_+^k$. We define the following classes of linear orders:

$$\mathcal{L}_1^{i+1}(\overline{c}) = \{\omega \cdot \mathbf{i} + L_{\overline{c}x}^i \mid x \in \mathbb{N}_+\}, \qquad \mathcal{L}_2^{i+1} = \{\omega \cdot \mathbf{i} + M \mid M \in \mathcal{M}^i\}, \qquad \mathcal{L}_3^{i+1} = \{\omega \cdot \mathbf{i} + K^i\}. \quad (5.13)$$

The linear orders $L_{\overline{c}}^{i+1}$, K^{i+1}, and M^{i+1} are defined as shuffle sums of unions of the above classes of linear orders:

$$L_{\overline{c}}^{i+1} = \mathrm{Shuf}(\mathcal{L}_1^{i+1}(\overline{c}) \cup \mathcal{L}_2^{i+1}), \qquad K^{i+1} = \mathrm{Shuf}(\mathcal{L}_2^{i+1} \cup \mathcal{L}_3^{i+1}), \qquad M^{i+1} = \mathrm{Shuf}(\mathcal{L}_2^{i+1}). \quad (5.14)$$

Recall that the set $\mathcal{M}^i$ is a singleton for $i > 1$, consisting of M^i. The next lemma can be proved similarly as Lemma 5.4.2.

Lemma 5.4.4 *Suppose $\mathcal{L}_1$ and $\mathcal{L}_2$ are two countable classes of linear orders such that each $L \in \mathcal{L}_1 \cup \mathcal{L}_2$ is isomorphic to a linear order of the form $\omega \cdot \mathbf{i} + K$, where $\omega \cdot \mathbf{i}$ is not isomorphic to any interval of K. Then*

$$\mathcal{L}_1 = \mathcal{L}_2 \iff \mathrm{Shuf}(\mathcal{L}_1) \cong \mathrm{Shuf}(\mathcal{L}_2) .$$

If Shuf($\mathcal{L}_1$) *contains an interval isomorphic to* $\omega \cdot (\mathbf{i}+\mathbf{1})$*, then there is a linear order* K *with* $\omega \cdot (\mathbf{i}+\mathbf{1}) + K \in \mathcal{L}_1$.

Proof. If $\mathcal{L}_1 = \mathcal{L}_2$, then it is clear that Shuf($\mathcal{L}_1$) $\cong$ Shuf($\mathcal{L}_2$). Conversely, suppose f is an isomorphism from Shuf($\mathcal{L}_1$) to Shuf($\mathcal{L}_2$). We prove that $\mathcal{L}_1 = \mathcal{L}_2$. By symmetry we only need to prove that $\mathcal{L}_1 \subseteq \mathcal{L}_2$.

Say $\mathcal{L}_j = \{L_{j,s} \mid s \in \mathbb{N}\}$ for $j \in \{1,2\}$. Intuitively, for $j \in \{1,2\}$, Shuf($\mathcal{L}_j$) can be viewed as obtained by replacing each $q \in \mathbb{Q}$ with a linear order $L(j,q) \cong L_{j,c(q)}$, where c is a dense $\mathbb{N}$-coloring. Fix $q \in \mathbb{Q}$. Suppose $f(L(1,q))$ contains elements in $L(2,p)$ and $L(2,p')$ for $p, p' \in \mathbb{Q}$ with $p < p'$. Then in $f(L(1,q))$ there are infinitely many disjoint intervals that are isomorphic to $\omega \cdot \mathbf{i}$, while in $L(1,q)$ there is exactly one such interval, a contradiction. Therefore f maps $L(1,q)$ into $L(2,p)$ for some $p \in \mathbb{Q}$.

If $f(L(1,q)) \subsetneq L(2,p)$, then $f^{-1}(L(2,p))$ contains an element $x \notin L(1,q)$. The argument from the previous paragraph with f replaced by f^{-1} again leads to a contradiction. Therefore $f(L(1,q)) = L(2,p)$. This means that for all $L \in \mathcal{L}_1$, there is $L' \in \mathcal{L}_2$ such that $L \cong L'$ and the lemma is proved.

Let $I \cong \omega \cdot (\mathbf{i}+\mathbf{1})$ be some interval in Shuf($\mathcal{L}_1$). First suppose there are $p < r$ in $\mathbb{Q}$ such that I intersects $L(1,p)$ and $L(1,r)$. But then $L(1,q) \subseteq I$ for all $q \in \{p+1, \ldots, r-1\}$, implying that $(\mathbb{Q}; \leq)$ embeds into $I \cong \omega \cdot (\mathbf{i}+\mathbf{1})$ which is impossible. Hence there is some $q \in \mathbb{Q}$ with $I \subseteq L(1,q) \in \mathcal{L}_1$. Then there is a linear order K such that $L(1,q) = \omega \cdot \mathbf{i} + K$. Since $\omega \cdot \mathbf{i}$ (let alone $\omega \cdot (\mathbf{i}+\mathbf{1})$) is no interval in K, the interval I has to intersect the initial segment $\omega \cdot \mathbf{i}$ of $L(1,q)$. But then ω has to be an initial segment of K, i.e., $L(1,q) = \omega \cdot (\mathbf{i}+\mathbf{1}) + K'$ for some linear order K'. ∎

Now notice that $\omega \cdot (\mathbf{i}+\mathbf{1})$ is not isomorphic to any interval of $L_{\overline{c}}^{i+1}$, K^{i+1}, or M^{i+1} (each of the orders $L_{\overline{c}x}^{i}$, K^i, and $M \in \mathcal{M}^i$ is a shuffle sum and therefore does not start with ω). Hence (P3) holds for $i+1$. Furthermore, the following holds:

$$
\begin{aligned}
P_{i+1}(\overline{c}) &\iff \exists x \in \mathbb{N}_+ : P_{i+1}(\overline{c}, x) \\
&\iff \exists x \in \mathbb{N}_+ : L_{\overline{c}x}^{i} \cong K^i \\
&\iff \mathcal{L}_3^{i+1} \subseteq \mathcal{L}_1^{i+1}(\overline{c}) \\
&\iff \mathcal{L}_1^{i+1}(\overline{c}) \cup \mathcal{L}_2^{i+1} = \mathcal{L}_2^{i+1} \cup \mathcal{L}_3^{i+1} \\
&\overset{\text{Lemma 5.4.4}}{\iff} L_{\overline{c}}^{i+1} \cong K^{i+1} \\
\neg P_{i+1}(\overline{c}) &\iff \forall x \in \mathbb{N}_+ : \neg P_{i+1}(\overline{c}, x) \\
&\iff \forall x \in \mathbb{N}_+ \; \exists M \in \mathcal{M}^i : L_{\overline{c}x}^{i} \cong M \\
&\iff \mathcal{L}_1^{i+1}(\overline{c}) \cup \mathcal{L}_2^{i+1} = \mathcal{L}_2^{i+1} \\
&\overset{\text{Lemma 5.4.4}}{\iff} L_{\overline{c}}^{i+1} \cong M^{i+1}
\end{aligned}
$$

We have shown (P1) and (P2) for $i+1$ in case i is odd.

5.4.1.3 Second induction step: from P_i to P_{i+1} for i even

Let $i \geq 1$ be even and consider the Π^0_{i+1}-predicate P_{i+1}. Again, we write k for $n-i$. For all $\overline{c} \in \mathbb{N}_+^k$, $P_{i+1}(\overline{c})$ is logically equivalent to $\forall x : P_i(\overline{c}, x)$. Since i is even, we must have $i \geq 2$. Therefore the set $\mathcal{M}^i$ is a singleton, consisting of the linear order M^i.

Fix $\overline{c} \in \mathbb{N}_+^k$. Define the classes of linear orders $\mathcal{L}_1^{i+1}(\overline{c})$, $\mathcal{L}_2^{i+1}$, and $\mathcal{L}_3^{i+1}$ using the same definition as in (5.13). The linear orders $L_{\overline{c}}^{i+1}$, K^{i+1}, and M^{i+1} are defined as follows:

$$L_{\overline{c}}^{i+1} = \mathrm{Shuf}(\mathcal{L}_1^{i+1}(\overline{c}) \cup \mathcal{L}_3^{i+1}), \qquad K^{i+1} = \mathrm{Shuf}(\mathcal{L}_3^{i+1}), \qquad M^{i+1} = \mathrm{Shuf}(\mathcal{L}_2^{i+1} \cup \mathcal{L}_3^{i+1}).$$

Again, $\omega \cdot (\mathbf{i}+\mathbf{1})$ is not isomorphic to any interval of $L_{\overline{c}}^{i+1}$, K^{i+1}, or M^{i+1}. Hence (P3) holds for $i+1$. Furthermore, the following holds:

$$\begin{aligned}
P_{i+1}(\overline{c}) &\iff \forall x \in \mathbb{N}_+ : P_i(\overline{c}, x) \\
&\iff \forall x \in \mathbb{N}_+ : L_{\overline{c}x}^{i} \cong K^i \\
&\iff \mathcal{L}_1^{i+1}(\overline{c}) \cup \mathcal{L}_3^{i+1} = \mathcal{L}_3^{i+1} \\
&\overset{\text{Lemma 5.4.4}}{\iff} L_{\overline{c}}^{i+1} \cong K^{i+1} \\
\neg P_{i+1}(\overline{c}) &\iff \exists x \in \mathbb{N}_+ : \neg P_i(\overline{c}, x) \\
&\iff \exists x \in \mathbb{N}_+ : L_{\overline{c}x}^{i} \cong M^i \\
&\iff \mathcal{L}_1^{i+1}(\overline{c}) \cup \mathcal{L}_3^{i+1} = \mathcal{L}_2^{i+1} \cup \mathcal{L}_3^{i+1} \\
&\overset{\text{Lemma 5.4.4}}{\iff} L_{\overline{c}}^{i+1} \cong M^{i+1}
\end{aligned}$$

We have shown (P1) and (P2) for $i+1$ in case i is even. This finishes the construction and proof for (P1), (P2), and (P3) in the inductive step.

5.4.2 Automaticity

To construct automatic presentations of the linear orders from the previous section, we first fix some notations. For $\overline{c} = (c_1, \ldots, c_k) \in \mathbb{N}_+^k$ and a symbol a, we re-define $a^{\overline{c}}$ as the word

$$a^{c_1}\sharp \cdots a^{c_k}\sharp \in \{a, \sharp\}^*.$$

Recall that Lemma 5.1.2 described a way to represent a polynomial $p(\overline{x}) \in \mathbb{N}[\overline{x}]$ in k variables using the number of accepting runs of an automaton $\mathcal{A}[p(\overline{x})]$. The next lemma re-states Lemma 5.1.2 with respect to the new definition of $a^{\overline{c}}$.

Lemma 5.4.5 *From a polynomial $p(\overline{x}) \in \mathbb{N}[\overline{x}]$ in k variables, one can effectively construct a non-deterministic automaton $\mathcal{A}[p(\overline{x})]$ on alphabet $\{a, \sharp\}$ such that $L(\mathcal{A}[p(\overline{x})]) = (a^+\sharp)^k$ and for all $\overline{c} \in \mathbb{N}_+^k$: $\mathcal{A}[p(\overline{x})]$ has exactly $p(\overline{c})$ accepting runs on input $a^{\overline{c}}$.*

Proof. We use the same proof as for Lemma 5.1.2. The only difference is when the polynomial $p(x_1, \ldots, x_k)$ is of the form x_i for some $i \in \{1, \ldots, k\}$. In this case, the automaton $\mathcal{A}[x_i]$ is (S, I, Δ, F) where $S = \{q_0, q_1, \ldots q_k, q_i'\}$, $I = \{q_0\}$, $F = \{q_k\}$ and the transition relation Δ is

$$\Delta = \{(q_{j-1}, \sharp, q_j) \mid 1 \leq j \leq k, j \neq i\} \cup \{(q, a, q) \mid q \in S\} \cup \{(q_{i-1}, a, q_i'), (q_i', \sharp, q_i)\}.$$

It is easy to see that $L(\mathcal{A}[x_i]) = (a^+\sharp)^k$ and $\mathcal{A}[x_i]$ has exactly c_i accepting runs on input $a^{\overline{c}}$ where $\overline{c} \in \mathbb{N}_+^k$. ∎

From now on, when referring to $\mathcal{A}[p(\overline{x})]$, we always assume it is defined in the sense of Lemma 5.4.5 (as opposed to Lemma 5.1.2). Let $\mathcal{A}$ be a non-deterministic finite automaton over the alphabet Σ and let Δ be the transition relation of $\mathcal{A}$. Recall the definition of the automaton $\mathrm{Run}_{\mathcal{A}}$ and the projection morphism $\pi : \Delta^* \to \Sigma^*$ from Section 5.1. Then, $\mathrm{Run}_{\mathcal{A}}$ is an automaton over the alphabet Δ. Assume that a lexicographic order $\leq_{\mathrm{lex}}$ has been defined on each of Σ^* and Δ^*. Define the automatic linear order $\sqsubseteq$ on $L(\mathrm{Run}_{\mathcal{A}})$ such that for all $w, w' \in L(\mathrm{Run}_{\mathcal{A}})$:

$$w \sqsubseteq w' \iff \pi(w) <_{\mathrm{lex}} \pi(w') \vee (\pi(w) = \pi(w') \wedge w \leq_{\mathrm{lex}} w'). \tag{5.15}$$

Let Σ_i be the alphabet $\{\sharp, \$_1, \ldots, \$_{i-1}, \$, 0, 1, a, b_1, b_2, b_3\}$. Fix the order $<$ on Σ_i such that

$$\$ < \$_1 < \cdots < \$_{i-1} < 0 < \sharp < a < b_1 < b_2 < b_3 < 1. \tag{5.16}$$

For any automaton $\mathcal{A}$ over Σ_i, fix an arbitrary order on the transition relation Δ of $\mathcal{A}$. Let $\leq_{\mathrm{lex}}$ be the lexicographic orders on Σ_i^* and Δ^* defined with respect to these orders, respectively. From now on, we will always let $\sqsubseteq$ be the linear order as defined in (5.15) with respect to $\leq_{\mathrm{lex}}$. For a regular language $L \subseteq \Sigma^*$ let $\mathrm{first}(L) = \{a \in \Sigma \mid \exists w \in \Sigma^* : aw \in L\}$. For $u \in \Sigma^*$, we use $L[u]$ to denote the language $u\Sigma^* \cap L$. Technically, in this section we prove by induction on i the following statement:

Proposition 5.4.6 *We can compute automata $\mathcal{A}^i$ over Σ_i such that:*

(1) $L(\mathcal{A}^1) = ((a^+\sharp)^n \cup b_1^+\sharp \cup b_2\sharp)\$R$ *for some regular language* $R \subseteq \Sigma_1^+$

(2) If $i > 1$, *then* $L(\mathcal{A}^i) = ((a^+\sharp)^{n-i+1} \cup b_1\sharp \cup b_2\sharp)\$R$ *for some regular language* $R \subseteq \Sigma_i^+$

(3) $L_{\overline{c}}^i \cong (\pi^{-1}(L(\mathcal{A}^i)[\overline{a}^{\overline{c}}]) \cap L(\mathrm{Run}_{\mathcal{A}^i}); \sqsubseteq)$ *for* $\overline{c} \in \mathbb{N}_+^{n-i+1}$

(4) $M_m^1 \cong (\pi^{-1}(L(\mathcal{A}^1)[b_1^m\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}^1}); \sqsubseteq)$ *for* $m \in \mathbb{N}_+$

(5) $M^i \cong (\pi^{-1}(L(\mathcal{A}^i)[b_1\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}^i}); \sqsubseteq)$ *for* $i > 1$

(6) $K^i \cong (\pi^{-1}(L(\mathcal{A}^i)[b_2\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}^i}); \sqsubseteq)$

Moreover, in (1) and (2) we have $\mathrm{first}(R) \subseteq \{0, 1\}$.

5.4.2.1 Effective automaticity of shuffle sums

This section shows that we can construct an automatic presentation of the shuffle sum of a class of automatic linear orders that are presented in some specific way. For a regular language D over an alphabet, which does neither contain 0 nor 1, let $\sigma(D) = (\{0, 1\}^*1D)^+$.

Lemma 5.4.7 *Let $\mathcal{A}$ be an automaton such that $L(\mathcal{A}) = ED\$F$ for regular languages $E, D \subseteq \{a, b_1, b_2, b_3, \sharp\}^*$ and $F \subseteq \Sigma_i^*$ (for some $i \in \{1, \ldots, n\}$). We can effectively compute an automaton $\sigma(\mathcal{A}, E)$ such that $L(\sigma(\mathcal{A}, E)) = E\$\sigma(D)\$F$ and for all $u \in E$:*

$$(\pi^{-1}(u\$\sigma(D)\$F) \cap L(\mathrm{Run}_{\sigma(\mathcal{A},E)}); \sqsubseteq) \cong \mathrm{Shuf}(\{(\pi^{-1}(uv\$F) \cap L(\mathrm{Run}_{\mathcal{A}}); \sqsubseteq) \mid v \in D\}).$$

Proof. Suppose $\mathcal{A} = (S, I, \Delta, S_f)$. Let $\Gamma = \{a, b_1, b_2, b_3, \sharp\}$. We first define the automaton

$$\mathcal{A}' = (S \times \{1, 2, \mathrm{loop}\}, I \times \{1\}, \Delta', S_f \times \{2\}).$$

The transition function Δ' of $\mathcal{A}'$ is defined as follows:

$$\begin{aligned}
\Delta' \quad = \quad & \{((q, 1), \alpha, (p, 1)) \mid (q, \alpha, p) \in \Delta, \alpha \in \Gamma\} \cup \\
& \{((q, 1), \$, (q, \mathrm{loop})) \mid q \in S\} \cup \\
& \{((q, \mathrm{loop}), \alpha, (q, \mathrm{loop})) \mid \alpha \in \Gamma \cup \{0, 1\}\} \cup \\
& \{((q, \mathrm{loop}), 1, (q, 2)) \mid q \in S\} \cup \\
& \{((q, 2), \alpha, (p, 2)) \mid (q, \alpha, p) \in \Delta\}
\end{aligned}$$

Intuitively, $\mathcal{A}'$ consists of two copies of $\mathcal{A}$ whose state spaces are $S \times \{1\}$ and $S \times \{2\}$. The automaton $\mathcal{A}'$ runs by starting simulating $\mathcal{A}$ on the first copy. When the first \$ is read, it stops the simulation. For this, the automaton stores the state q by moving to the "looping state" (q, loop). The automaton will stay in (q, loop) unless 1 is read, in which case, it may "guess" that it reads the last 1 before the second \$ in the input. If so, it goes out of (q, loop) and continues the simulation in the second copy of $\mathcal{A}$ and accepts the input word if the run stops at a final state. If the guess was not correct and there is another 1 before the second \$ in the input, then the run will necessarily reject.

It is easy to see that for all $u_1, u_2 \in \Gamma^*$, $v \in (\Gamma \cup \{0, 1\})^*1$ and $u_3 \in F$, the number of accepting runs of $\mathcal{A}'$ on $u_1\$vu_2\$u_3$ is the same as the number of accepting runs of $\mathcal{A}$ on

$u_1u_2\$u_3$, i.e.,

$$|L(\mathrm{Run}_{\mathcal{A}'}) \cap \pi^{-1}(u_1\$vu_2\$u_3)| = |L(\mathrm{Run}_{\mathcal{A}}) \cap \pi^{-1}(u_1u_2\$u_3)|. \tag{5.17}$$

Let

$$\sigma(\mathcal{A}, E) \;=\; E\$\sigma(D)\$F \;\cap\; \mathcal{A}'.$$

Note that $L(\sigma(\mathcal{A}, E)) = E\$\sigma(D)\$F$. Also, for any $u_1 \in E$, $v \in (\{0,1\}^*1D)^*\{0,1\}^*1$, $u_2 \in D$, and $u_3 \in F$, the number of accepting runs of $\sigma(\mathcal{A}, E)$ on $u_1\$vu_2\$u_3$ equals the number of accepting runs of $\mathcal{A}'$ on $u_1\$vu_2\$u_3$, which is, by (5.17), equal to the number of accepting runs of $\mathcal{A}$ on $u_1u_2\$u_3$. Hence, we have

$$|L(\mathrm{Run}_{\sigma(\mathcal{A},E)}) \cap \pi^{-1}(u_1\$vu_2\$u_3)| = |L(\mathrm{Run}_{\mathcal{A}}) \cap \pi^{-1}(u_1u_2\$u_3)|. \tag{5.18}$$

We prove the following claim.

Claim 1. For all $u_1 \in E$, $v \in (\{0,1\}^*1D)^*\{0,1\}^*1$ and $u_2 \in D$,

$$(\pi^{-1}(u_1\$vu_2\$F) \cap L(\mathrm{Run}_{\sigma(\mathcal{A},E)}); \sqsubseteq) \cong (\pi^{-1}(u_1u_2\$F) \cap L(\mathrm{Run}_{\mathcal{A}}); \sqsubseteq). \tag{5.19}$$

For $u \in F$, let $L(u) = (\pi^{-1}(u_1u_2\$u) \cap L(\mathrm{Run}_{\mathcal{A}}); \sqsubseteq)$. Note that this is a finite linear order. Consider the linear order $(F; \leq_{\mathrm{lex}})$. By definition of $\sqsubseteq$,

$$(\pi^{-1}(u_1u_2\$F) \cap L(\mathrm{Run}_{\mathcal{A}}); \sqsubseteq) \cong \sum_{u \in F} L(u).$$

By (5.18), $L(u) \cong (\pi^{-1}(u_1\$vu_2\$u) \cap L(\mathrm{Run}_{\sigma(\mathcal{A},E)}); \sqsubseteq)$. By definition of $\sqsubseteq$ again,

$$\begin{aligned}(\pi^{-1}(u_1\$vu_2\$F) \cap L(\mathrm{Run}_{\sigma(\mathcal{A},E)}); \sqsubseteq) &\cong \sum_{u \in F} (\pi^{-1}(u_1\$vu_2\$u) \cap L(\mathrm{Run}_{\sigma(\mathcal{A},E)}); \sqsubseteq) \\ &\cong \sum_{u \in F} L(u) \\ &\cong (\pi^{-1}(u_1u_2\$F) \cap L(\mathrm{Run}_{\mathcal{A}}); \sqsubseteq).\end{aligned}$$

This proves Claim 1.

Let $c : \sigma(D) \to D$ be the function such that

$$\forall x \in (\{0,1\}^*1D)^*\{0,1\}^*1 \; \forall u \in D : \; c(xu) = u.$$

Claim 2. $(\sigma(D); \leq_{\mathrm{lex}}) \cong (\mathbb{Q}; \leq)$ and the function c is a dense D-coloring of $(\sigma(D); \leq_{\mathrm{lex}})$.

First, for every $w = x1u \in \sigma(D)$ with $x \in (\{0,1\}^*1D)\{0,1\}^*$ and $u \in D$, we have

$$x01u <_{\mathrm{lex}} w <_{\mathrm{lex}} x11u.$$

Hence, $(\sigma(D); \leq_{\text{lex}})$ does not have a smallest or largest element. It remains to show that the linear order $(\sigma(D); \leq_{\text{lex}})$ is densely D-colored by c (this implies that $(\sigma(D); \leq_{\text{lex}})$ is dense and hence, by Cantor's theorem, isomorphic to $(\mathbb{Q}; \leq)$). Consider two words $w_1, w_2 \in \sigma(D)$ such that $w_1 <_{\text{lex}} w_2$. There are two cases.

Case 1. $w_1 = x\alpha y$, $w_2 = x\beta z$ for $x, y, z \in (\Gamma \cup \{0,1\})^*$ and $\alpha, \beta \in \Gamma \cup \{0,1\}$ such that $\alpha < \beta$. In this case, for all $u \in D$, we have

$$w_1 <_{\text{lex}} w_1 1u <_{\text{lex}} w_2 \quad \text{and} \quad w_1 1u \in \sigma(D).$$

Case 2. $w_2 = w_1 x$ for some $x \in (\Gamma \cup \{0,1\})^+$. Since $w_2 \in \sigma(D)$, we have $x \notin 0^*$. Say $x = 0^j \alpha y$ for some $j \geq 0$, $\alpha \neq 0$ and $y \in (\Gamma \cup \{0,1\})^*$. We must have $\alpha \in \{1, a, b_1, b_2, b_3, \sharp\}$. Since every symbol from this set is larger than 0 (see (5.16)) we must have $\alpha > 0$. Then for all $u \in D$, we have

$$w_1 <_{\text{lex}} w_1 0^{j+1} 1u <_{\text{lex}} w_2 \quad \text{and} \quad w_1 0^{j+1} 1u \in \sigma(D).$$

Hence $(\sigma(D); \leq_{\text{lex}})$ is indeed densely colored by c. This proves Claim 2.

Since \$ is the minimum in the order $<$ on Σ_i, for any $u \in E$, $v, v' \in \sigma(D)$ and $w, w' \in F$, we have

$$v <_{\text{lex}} v' \implies u\$v\$w <_{\text{lex}} u\$v'\$w'.$$

Therefore,

$$\begin{aligned}
(\pi^{-1}(u\$\sigma(D)\$F) \cap L(\text{Run}_{\sigma(\mathcal{A},E)}); \sqsubseteq) &\cong \sum_{v \in \sigma(D)} (\pi^{-1}(u\$v\$F) \cap L(\text{Run}_{\sigma(\mathcal{A},E)}); \sqsubseteq) \\
&\overset{\text{Claim 1}}{\cong} \sum_{v \in \sigma(D)} (\pi^{-1}(uc(v)\$F) \cap L(\text{Run}_{\mathcal{A}}); \sqsubseteq) \\
&\overset{\text{Claim 2}}{\cong} \text{Shuf}(\{(\pi^{-1}(uv\$F) \cap L(\text{Run}_{\mathcal{A}}); \sqsubseteq) \mid v \in D\}).
\end{aligned}$$

∎

5.4.2.2 Base case: automatic presentations for $L^1_{\frac{1}{c}}$, K^1, and M^1_m

Recall the notations from Section 5.4.1.1. In the following, if D is a regular language and $\mathcal{A}$ is a finite non-deterministic automaton then we denote by $D\mathcal{A}$ a finite automaton that results from the disjoint union of a deterministic automaton $\mathcal{A}_D$ for D and the automaton $\mathcal{A}$ by adding all transitions (q, a, p) where: (i) q is a state of $\mathcal{A}_D$, (ii) there is a transition (q, a, q') in $\mathcal{A}_D$, where q' is a final state of $\mathcal{A}_D$, and (iii) p is an initial state of $\mathcal{A}$. Clearly, $L(D\mathcal{A}) = DL(\mathcal{A})$. We will only apply this definition in case the product $DL(A)$ is unambiguous. This means that if $u \in DL(A)$ then there exists a unique factorization $u = u_1 u_2$ with $u_1 \in D$ and $u_2 \in L(A)$. The following lemma is easy to prove:

Lemma 5.4.8 *Let $\mathcal{A}$ be a finite non-deterministic automaton and let D be a regular language such that the product $DL(A)$ is unambiguous. Let $u_1 \in D$ and $u_2 \in L(\mathcal{A})$. Then, the number of accepting runs of $D\mathcal{A}$ on u_1u_2 equals the number of accepting runs of $\mathcal{A}$ on u_2.*

Lemma 5.4.9 *From two given polynomials $q_1(\overline{x}), q_2(\overline{x}) \in \mathbb{N}[\overline{x}]$ in k variables, one can effectively construct an automaton $\mathcal{A}[q_1, q_2]$ over the alphabet $\{a, \sharp, \$\}$ such that*

- *$L(\mathcal{A}[q_1, q_2]) = (a^+\sharp)^k\$$ and*
- *For all $\overline{c} \in \mathbb{N}_+^k$, $(\pi^{-1}(a^{\overline{c}}\$) \cap L(\mathrm{Run}_{\mathcal{A}[q_1,q_2]}); \sqsubseteq) \cong L[q_1(\overline{c}), q_2(\overline{c})]$.*

Proof. We construct $\mathcal{A}[q_1, q_2]$ by taking a copy of $\mathcal{A}[C(q_1(\overline{x}), q_2(\overline{x}))]$ (see Lemma 5.4.5), adding a new state $q_\$$ and transitions $(q_f, \$, q_\$)$ for each accepting state q_f in $\mathcal{A}[C(q_1(\overline{x}), q_2(\overline{x}))]$ and making $q_\$$ the only accepting state of $\mathcal{A}[q_1, q_2]$. Note that for any $\overline{c} \in \mathbb{N}_+^k$, the number of accepting runs of $\mathcal{A}[q_1, q_2]$ on $a^{\overline{c}}\$$ is the same as the number of accepting runs of $\mathcal{A}[C(q_1(\overline{x}), q_2(\overline{x}))]$ on $a^{\overline{c}}$, which is equal to $C(q_1(\overline{c}), q_2(\overline{c}))$. Hence, $(\pi^{-1}(a^{\overline{c}}\$) \cap L(\mathrm{Run}_{\mathcal{A}[q_1,q_2]}); \sqsubseteq)$ forms a copy of $L[q_1(\overline{c}), q_2(\overline{c})]$ and the lemma is proved. ∎

By Lemma 5.4.9, we can construct automata $\mathcal{A}_1 = \mathcal{A}[p_1(\overline{x}) + x_{\ell+1}, p_2(\overline{x}) + x_{\ell+1}]$, where $\overline{x} \in \mathbb{N}_+^\ell$, over the alphabet $\{a, \sharp, \$\}$, $\mathcal{A}_2 = \mathcal{A}[x_1 + x_2, x_1 + x_2]$ over the alphabet $\{b_1, \sharp, \$\}$, $\mathcal{A}_3 = \mathcal{A}[x_1 + x_2, x_1]$ over the alphabet $\{b_2, \sharp, \$\}$ and $\mathcal{A}_4 = \mathcal{A}[x_1, x_1 + x_2]$ over the alphabet $\{b_3, \sharp, \$\}$ such that:

$$\forall \overline{c} \in \mathbb{N}_+^\ell\ \forall c_{\ell+1} \in \mathbb{N}_+ : (\pi^{-1}(a^{\overline{c}c_{\ell+1}}\$) \cap L(\mathrm{Run}_{\mathcal{A}_1}); \sqsubseteq) \cong L[p_1(\overline{c}) + c_{\ell+1}, p_2(\overline{c}) + c_{\ell+1}] \quad (5.20)$$

$$\forall e_1, e_2 \in \mathbb{N}_+ : (\pi^{-1}(b_1^{e_1}\sharp b_1^{e_2}\sharp\$) \cap L(\mathrm{Run}_{\mathcal{A}_2}); \sqsubseteq) \cong L[e_1 + e_2, e_1 + e_2] \quad (5.21)$$

$$\forall e_1, e_2 \in \mathbb{N}_+ : (\pi^{-1}(b_2^{e_1}\sharp b_2^{e_2}\sharp\$) \cap L(\mathrm{Run}_{\mathcal{A}_3}); \sqsubseteq) \cong L[e_1 + e_2, e_1] \quad (5.22)$$

$$\forall e_1, e_2 \in \mathbb{N}_+ : (\pi^{-1}(b_3^{e_1}\sharp b_3^{e_2}\sharp\$) \cap L(\mathrm{Run}_{\mathcal{A}_4}); \sqsubseteq) \cong L[e_1, e_1 + e_2] \quad (5.23)$$

Define the following automata:

$$\mathcal{A}_1^0 = \mathcal{A}_1 \uplus ((a^+\sharp)^n(\mathcal{A}_3 \uplus \mathcal{A}_4)), \qquad \mathcal{A}_2^0 = \mathcal{A}_2 \uplus (b_1^+\sharp(\mathcal{A}_3 \uplus \mathcal{A}_4)), \qquad \mathcal{A}_3^0 = b_2\sharp(\mathcal{A}_3 \uplus \mathcal{A}_4).$$

Note that

$$L(\mathcal{A}_1^0) = (a^+\sharp)^n\Big((a^+\sharp)^{\ell-n+1} \cup (b_2^+\sharp)^2 \cup (b_3^+\sharp)^2\Big)\$,$$

$$L(\mathcal{A}_2^0) = b_1^+\sharp\Big(b_1^+\sharp \cup (b_2^+\sharp)^2 \cup (b_3^+\sharp)^2\Big)\$,$$

$$L(\mathcal{A}_3^0) = b_2\sharp\Big((b_2^+\sharp)^2 \cup (b_3^+\sharp)^2\Big)\$.$$

Hence, applying Lemma 5.4.7 (with $F = \{\varepsilon\}$), we can effectively construct automata $\mathcal{A}^1_j$ ($j \in \{1,2,3\}$) as follows:

$$\mathcal{A}^1_1 = \sigma(\mathcal{A}^0_1, (a^+\sharp)^n), \qquad \mathcal{A}^1_2 = \sigma(\mathcal{A}^0_2, b_1^+\sharp), \qquad \mathcal{A}^1_3 = \sigma(\mathcal{A}^0_3, b_2\sharp).$$

For all $\overline{c} \in \mathbb{N}_+^n$ we get:

$$\begin{aligned}
&(\pi^{-1}(L(\mathcal{A}^1_1)[a^{\overline{c}}]) \cap L(\mathrm{Run}_{\mathcal{A}^1_1}); \sqsubseteq) \overset{\text{Lemma 5.4.7}}{\cong} \\
&\quad \mathrm{Shuf}(\{(\pi^{-1}(a^{\overline{c}}v\$) \cap L(\mathrm{Run}_{\mathcal{A}^0_1}); \sqsubseteq) \mid v \in (a^+\sharp)^{\ell-n+1} \cup (b_2^+\sharp)^2 \cup (b_3^+\sharp)^2\}) = \\
&\quad \mathrm{Shuf}(\{(\pi^{-1}(a^{\overline{c}\,\overline{e}}\$) \cap L(\mathrm{Run}_{\mathcal{A}^0_1}); \sqsubseteq) \mid \overline{e} \in \mathbb{N}_+^{\ell-n+1}\} \cup \\
&\qquad \{(\pi^{-1}(a^{\overline{c}}b_2^{e_1}\sharp b_2^{e_2}\sharp\$) \cap L(\mathrm{Run}_{\mathcal{A}^0_1}); \sqsubseteq) \mid e_1, e_2 \in \mathbb{N}_+\} \cup \\
&\qquad \{(\pi^{-1}(a^{\overline{c}}b_3^{e_1}\sharp b_3^{e_2}\sharp\$) \cap L(\mathrm{Run}_{\mathcal{A}^0_1}); \sqsubseteq) \mid e_1, e_2 \in \mathbb{N}_+\}) \overset{\text{Lemma 5.4.8}}{\cong} \\
&\quad \mathrm{Shuf}(\{(\pi^{-1}(a^{\overline{c}\,\overline{e}}\$) \cap L(\mathrm{Run}_{\mathcal{A}_1}); \sqsubseteq) \mid \overline{e} \in \mathbb{N}_+^{\ell-n+1}\} \cup \\
&\qquad \{(\pi^{-1}(b_2^{e_1}\sharp b_2^{e_2}\sharp\$) \cap L(\mathrm{Run}_{\mathcal{A}_3}); \sqsubseteq) \mid e_1, e_2 \in \mathbb{N}_+\} \cup \\
&\qquad \{(\pi^{-1}(b_3^{e_1}\sharp b_3^{e_2}\sharp\$) \cap L(\mathrm{Run}_{\mathcal{A}_4}); \sqsubseteq) \mid e_1, e_2 \in \mathbb{N}_+\}) \overset{(5.20)\text{–}(5.23)}{=} \\
&\quad \mathrm{Shuf}(\{L[p_1(\overline{c},\overline{e}) + e_{\ell+1}, p_2(\overline{c},\overline{e}) + e_{\ell+1}] \mid \overline{e} \in \mathbb{N}_+^{\ell-n}, e_{\ell+1} \in \mathbb{N}_+\} \cup \\
&\qquad \{L[e_1 + e_2, e_1] \mid e_1, e_2 \in \mathbb{N}_+\} \cup \{L[e_1, e_1 + e_2] \mid e_1, e_2 \in \mathbb{N}_+\}) \overset{(5.9)\text{–}(5.12)}{=} \\
&\quad \mathrm{Shuf}(\mathcal{L}^1_1(\overline{c}) \cup \mathcal{L}^1_3 \cup \mathcal{L}^1_4) \cong L^1_{\overline{c}}
\end{aligned}$$

Similar calculations yield:

$$\begin{aligned}
\forall m \in \mathbb{N}_+ : (\pi^{-1}(L(\mathcal{A}^1_2)[b_1^m\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}^1_2}); \sqsubseteq) &\cong \mathrm{Shuf}(\mathcal{L}^1_2(m) \cup \mathcal{L}^1_3 \cup \mathcal{L}^1_4) \cong M^1_m \\
(\pi^{-1}(L(\mathcal{A}^1_3)[b_2\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}^1_3}); \sqsubseteq) &\cong \mathrm{Shuf}(\mathcal{L}^1_3 \cup \mathcal{L}^1_4) \cong K^1
\end{aligned}$$

Let $\mathcal{A}^1 = \mathcal{A}^1_1 \uplus \mathcal{A}^1_2 \uplus \mathcal{A}^1_3$. It is easy to see that $L(\mathcal{A}^1) = ((a^+\sharp)^n \cup b_1^+\sharp \cup b_2\sharp)\$R$ for some regular language $R \subseteq \Sigma_1^+$ with $\mathrm{first}(R) \subseteq \{0,1\}$. Hence $\mathcal{A}^1$ satisfies the statement in Proposition 5.4.6.

5.4.2.3 First inductive step: automatic presentations for $L^{i+1}_{\overline{c}}$, K^{i+1}, M^{i+1} for i odd

Let $i \geq 1$ be an odd number. Recall the notations from Section 5.4.1.2. We write k for $n - i$. By applying the inductive assumption, we obtain an automaton $\mathcal{A}^i$ such that $L(\mathcal{A}^i) = ((a^+\sharp)^{k+1} \cup \beta\sharp \cup b_2\sharp)\$R$ for some regular language $R \subseteq \Sigma_i^*$ where $\beta = b_1^+$ if $i = 1$, and $\beta = b_1$

otherwise. Furthermore, first$(R) \subseteq \{0,1\}$ and the following hold for $\mathcal{A}^i$:

$$\forall \bar{c} \in \mathbb{N}_+^{k+1} : L_{\bar{c}}^i \cong (\pi^{-1}(a^{\bar{c}}\$R) \cap L(\mathrm{Run}_{\mathcal{A}^i}); \sqsubseteq) \tag{5.24}$$

$$\mathcal{M}^i \cong \{(\pi^{-1}(u\sharp\$R) \cap L(\mathrm{Run}_{\mathcal{A}^i}); \sqsubseteq) \mid u \in \beta\} \tag{5.25}$$

$$K^i \cong (\pi^{-1}(b_2\sharp\$R) \cap L(\mathrm{Run}_{\mathcal{A}^i}); \sqsubseteq) \tag{5.26}$$

For any $1 \leq j \leq n$, let $S_j = \$_1^+ \cup \cdots \cup \$_j^+$. It is easy to see that

$$(S_j; \leq_{\mathrm{lex}}) \cong \omega \cdot \mathbf{j}. \tag{5.27}$$

Define the automata $\mathcal{B}_1^i$, $\mathcal{B}_2^i$, and $\mathcal{B}_3^i$ as

$$\mathcal{B}_1^i = ((a^+\sharp)^{k+1}\$R \cap \mathcal{A}^i) \uplus (a^+\sharp)^{k+1}\$S_i, \tag{5.28}$$

$$\mathcal{B}_2^i = (\beta\sharp\$R \cap \mathcal{A}^i) \uplus \beta\sharp\$S_i, \tag{5.29}$$

$$\mathcal{B}_3^i = (b_2\sharp\$R \cap \mathcal{A}^i) \uplus b_2\sharp\$S_i. \tag{5.30}$$

By (5.16), (5.24)–(5.27), and the fact that first$(R) \subseteq \{0,1\}$, we have

$$\forall \bar{c} \in \mathbb{N}_+^{k+1} : (\pi^{-1}(a^{\bar{c}}\$(S_i \cup R)) \cap L(\mathrm{Run}_{\mathcal{B}_1^i}); \sqsubseteq) \cong \omega \cdot \mathbf{i} + L_{\bar{c}}^i, \tag{5.31}$$

$$\{(\pi^{-1}(u\sharp\$(S_i \cup R)) \cap L(\mathrm{Run}_{\mathcal{B}_2^i}); \sqsubseteq) \mid u \in \beta\} \cong \{\omega \cdot \mathbf{i} + M \mid M \in \mathcal{M}^i\}, \tag{5.32}$$

$$(\pi^{-1}(b_2\sharp\$(S_i \cup R)) \cap L(\mathrm{Run}_{\mathcal{B}_3^i}); \sqsubseteq) \cong \omega \cdot \mathbf{i} + K^i. \tag{5.33}$$

Now construct the automata C_1^i, C_2^i, and C_3^i as follows:

$$C_1^i = \mathcal{B}_1^i \uplus (a^+\sharp)^k\mathcal{B}_2^i, \qquad C_2^i = b_1\sharp\mathcal{B}_2^i, \qquad C_3^i = b_2\sharp(\mathcal{B}_2^i \uplus \mathcal{B}_3^i).$$

We have

$$\begin{aligned} L(C_1^i) &= (a^+\sharp)^k(a^+\sharp \cup \beta\sharp)\$(S_i \cup R), \\ L(C_2^i) &= b_1\sharp\beta\sharp\$(S_i \cup R), \\ L(C_3^i) &= b_2\sharp(\beta\sharp \cup b_2\sharp)\$(S_i \cup R). \end{aligned}$$

Hence, we can apply Lemma 5.4.7 to C_1^i, C_2^i, and C_3^i (with $F = S_i \cup R$) to define the following automata:

$$\mathcal{A}_1^{i+1} = \sigma(C_1^i, (a^+\sharp)^k), \qquad \mathcal{A}_2^{i+1} = \sigma(C_2^i, b_1\sharp), \qquad \mathcal{A}_3^{i+1} = \sigma(C_3^i, b_2\sharp).$$

For all $\overline{c} \in \mathbb{N}_+^k$ we get:

$$
\begin{aligned}
&(\pi^{-1}(L(\mathcal{A}_1^{i+1})[a^{\overline{c}}]) \cap L(\mathrm{Run}_{\mathcal{A}_1^{i+1}}); \sqsubseteq) \overset{\text{Lemma 5.4.7}}{\cong} \\
&\quad \mathrm{Shuf}(\{(\pi^{-1}(a^{\overline{c}} v\$(S_i \cup R)) \cap L(\mathrm{Run}_{C_1^i}); \sqsubseteq) \mid v \in a^+\sharp \cup \beta\sharp\}) = \\
&\quad \mathrm{Shuf}(\{(\pi^{-1}(a^{\overline{c}} e\$(S_i \cup R)) \cap L(\mathrm{Run}_{C_1^i}); \sqsubseteq) \mid e \in \mathbb{N}_+\} \cup \\
&\qquad \{(\pi^{-1}(a^{\overline{c}} u\sharp\$(S_i \cup R)) \cap L(\mathrm{Run}_{C_1^i}); \sqsubseteq) \mid u \in \beta\}) \overset{\text{Lemma 5.4.8}}{\cong} \\
&\quad \mathrm{Shuf}(\{(\pi^{-1}(a^{\overline{c}} e\$(S_i \cup R)) \cap L(\mathrm{Run}_{\mathcal{B}_1^i}); \sqsubseteq) \mid e \in \mathbb{N}_+\} \cup \\
&\qquad \{(\pi^{-1}(u\sharp\$(S_i \cup R)) \cap L(\mathrm{Run}_{\mathcal{B}_2^i}); \sqsubseteq) \mid u \in \beta\}) \overset{(5.31),\,(5.32)}{=} \\
&\quad \mathrm{Shuf}(\{\omega \cdot \mathbf{i} + \mathcal{L}_{\overline{c}e}^i \mid e \in \mathbb{N}_+\} \cup \{\omega \cdot \mathbf{i} + M \mid M \in \mathcal{M}^i\}) \overset{(5.13),\,(5.14)}{\cong} L_{\overline{c}}^{i+1}
\end{aligned}
$$

Similarly, we can show:

$$
\begin{aligned}
(\pi^{-1}(L(\mathcal{A}_2^{i+1})[b_1\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}_2^{i+1}}); \sqsubseteq) &\cong \mathrm{Shuf}(\{\omega \cdot \mathbf{i} + M \mid M \in \mathcal{M}^i\}) \cong M^{i+1}, \\
(\pi^{-1}(L(\mathcal{A}_3^{i+1})[b_2\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}_3^{i+1}}); \sqsubseteq) &\cong \mathrm{Shuf}(\{\omega \cdot \mathbf{i} + M \mid M \in \mathcal{M}^i\} \cup \{\omega \cdot \mathbf{i} + K^i\}) \cong K^{i+1}.
\end{aligned}
$$

Let $\mathcal{A}^{i+1} = \mathcal{A}_1^{i+1} \uplus \mathcal{A}_2^{i+1} \uplus \mathcal{A}_3^{i+1}$. It is easy to see that $L(\mathcal{A}^{i+1}) = ((a^+\sharp)^k \cup b_1\sharp \cup b_2\sharp)\$R'$ for some regular language $R' \subseteq \Sigma_{i+1}^+$ with $\mathrm{first}(R') \subseteq \{0, 1\}$. Hence $\mathcal{A}^{i+1}$ satisfies the statement in Proposition 5.4.6.

5.4.2.4 Second inductive step: automatic presentations for $L_{\overline{c}}^{i+1}$, K^{i+1}, M^{i+1} for i even

Using the same technique, we can construct automatic presentations for $L_{\overline{c}}^{i+1}$ ($\overline{c} \in \mathbb{N}_+^k$), M^{i+1}, and K^{i+1} in case i is even. We first define the automata $\mathcal{B}_1^i$, $\mathcal{B}_2^i$, and $\mathcal{B}_3^i$ as in (5.28)–(5.30), with $\beta = b_1$ this time. Then we construct

$$C_1^i = \mathcal{B}_1^i \uplus (a^+\sharp)^k \mathcal{B}_3^i, \qquad C_2^i = b_1\sharp(\mathcal{B}_2^i \uplus \mathcal{B}_3^i), \qquad C_3^i = b_2\sharp\mathcal{B}_3^i.$$

We define the following automata by applying Lemma 5.4.7:

$$\mathcal{A}_1^{i+1} = \sigma(C_1^i, (a^+\sharp)^k), \qquad \mathcal{A}_2^{i+1} = \sigma(C_2^i, b_1\sharp), \qquad \mathcal{A}_3^{i+1} = \sigma(C_3^i, b_2\sharp).$$

By Lemma 5.4.7, it is easy to check the following:

$$\begin{aligned}
\forall \bar{c} \in \mathbb{N}_+^k : (\pi^{-1}(L(\mathcal{A}_1^{i+1})[a^{\bar{c}}]) \cap L(\mathrm{Run}_{\mathcal{A}_1^{i+1}}); \sqsubseteq) &\cong \mathrm{Shuf}(\{\omega \cdot \mathbf{i} + \mathcal{L}^i_{\bar{c}x} \mid x \in \mathbb{N}_+\} \cup \{\omega \cdot \mathbf{i} + K^i\}) \\
&\cong L_{\bar{c}}^{i+1}, \\
(\pi^{-1}(L(\mathcal{A}_2^{i+1})[b_1\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}_2^{i+1}}); \sqsubseteq) &\cong \mathrm{Shuf}(\{\omega \cdot \mathbf{i} + M^i\} \cup \{\omega \cdot \mathbf{i} + K^i\}) \\
&\cong M^{i+1}, \\
(\pi^{-1}(L(\mathcal{A}_3^{i+1})[b_2\sharp]) \cap L(\mathrm{Run}_{\mathcal{A}_3^{i+1}}); \sqsubseteq) &\cong \mathrm{Shuf}(\{\omega \cdot \mathbf{i} + K^i\}) \\
&\cong K^{i+1}.
\end{aligned}$$

Let $\mathcal{A}^{i+1} = \mathcal{A}_1^{i+1} \uplus \mathcal{A}_2^{i+1} \uplus \mathcal{A}_3^{i+1}$. It is easy to see that $L(\mathcal{A}^{i+1}) \subseteq ((a^+\sharp)^k \cup b_1\sharp \cup b_2\sharp)\$R'$ for some regular language $R' \subseteq \Sigma_{i+1}^+$ with $\mathrm{first}(R') \subseteq \{0,1\}$. Hence $\mathcal{A}^{i+1}$ satisfies the statement in Proposition 5.4.6. This finishes the construction in the inductive step and hence the proof of Proposition 5.4.6. Hence we obtain:

Theorem 5.4.10 *The isomorphism problem for the class of automatic linear orders is at least as hard as* $\mathsf{FOTh}(\mathbb{N}; +, \times)$.

In [77], it is shown that every linear order has finite FC-rank. We do not define the FC-rank of a linear order in general, see e.g. [77]. A linear order $(L, \leq)$ has FC-rank 1, if after identifying all $x, y \in L$ such that the interval $[x, y]$ is finite, one obtains a dense ordering or the singleton linear order. The result of [77] mentioned above suggests that the isomorphism problem might be simpler for linear orders of low FC-rank. We now prove that this is not the case:

Corollary 5.4.11 *The isomorphism problem for automatic linear orders of FC-rank 1 is at least as* $\mathsf{FOTh}(\mathbb{N}; +, \times)$.

Proof. We provide a reduction from the isomorphism problem for automatic linear orders (of arbitrary rank): if $(L, \leq)$ is an automatic linear order, then so is $(K, \leq) = ((-1, 0] + [1, 2)) \cdot (L, \leq)$ (this linear order is obtained from L by replacing each point with a copy of the rational numbers in $(-1, 0] \cup [1, 2)$). Then $(K, \leq)$ has FC-rank 1: Only the copies of 0 and 1 will be identified, and the resulting order is isomorphic to $(\mathbb{Q}, \leq)$. Moreover, $(L, \leq)$ is isomorphic to the set of all $x \in K$ satisfying $\exists z > x \; \forall y : (x < y \leq z \to y = z)$. Hence $(L, \leq) \cong (L', \leq')$ if and only if $((-1, 0] + [1, 2)) \cdot (L, \leq) \cong ((-1, 0] + [1, 2)) \cdot (L', \leq')$, which completes the reduction. ∎

5.5 Arithmetical isomorphisms

We conclude this paper with an application of Theorem 5.2.13 and Theorem 5.4.10. The following corollary shows that although automatic structures look simple (especially for

automatic trees), there may be no "simple" isomorphism between two automatic copies of the same structure. An isomorphism f between two automatic structures with domains L_1 and L_2, resp., is a Σ^0_k-isomorphism, if the set $\{(x, f(x)) \mid x \in L_1\}$ belongs to Σ^0_k.

Corollary 5.5.1 *For any $k \in \mathbb{N}$, there exist two isomorphic automatic trees of finite height (and two automatic linear orders) without any Σ^0_k-isomorphism.*

Proof. Let $T_1 = (D_1; E_1)$ and $T_2 = (D_2; E_2)$ be two automatic trees. Let $P_1(x, y), P_2(x, y), \ldots$ be an effective enumeration of all binary Σ^0_k-predicates. This means that from given $e \geq 1$ we can effectively compute a description (e.g. a Σ_k-formula over $(\mathbb{N}; +, \times)$) of the predicate $P_e(x, y)$. We define the statement iso(T_1, T_2, k) as follows:

$$\exists e\, \forall x_1, x_2 \in D_1\, \exists y_1, y_2 \in D_2 : P_e(x_1, y_1) \ \wedge\ P_e(x_2, y_2) \wedge \\ (x_1 = x_2 \leftrightarrow y_1 = y_2) \ \wedge\ ((x_1, x_2) \in E_1 \leftrightarrow (y_1, y_2) \in E_2).$$

Since P_e is a Σ^0_k-predicate, this is a Σ^0_{k+2}-statement, which expresses the existence of a Σ^0_k-isomorphism from T_1 to T_2.

By Theorem 4.5.9, there is a natural number n such that the isomorphism problem on the class $\mathcal{T}_n$ of automatic trees of height at most n is Σ_{k+3}-hard. If for all $T_1, T_2 \in \mathcal{T}_n$ with $T_1 \cong T_2$ there exists a Σ^0_k-isomorphism from T_1 to T_2, then the isomorphism problem on $\mathcal{T}_n$ reduces to checking existence of a Σ^0_k-isomorphism, which is in Σ^0_{k+2} by the above consideration. Hence, there must be $T_1, T_2 \in \mathcal{T}_n$ with $T_1 \cong T_2$ but there is no Σ^0_k-isomorphism between them.

The corollary for linear orders can be proved in the same way, where in the definition of iso(T_1, T_2, k) we replace $(x_1, x_2) \in E_1 \leftrightarrow (y_1, y_2) \in E_2$ with $x_1 <_1 x_2 \leftrightarrow y_1 <_2 y_2$, where $<_1$ and $<_2$ are the linear orders of T_1 and T_2, respectively. ∎

Chapter 6

Computably Categorical Graphs with Finite Components

In this chapter we study the computable categoricity for graphs, the graphs with exactly one computable isomorphism type. We focus on the class of strongly locally finite graphs; recall that all components of these graphs are finite. By Corollary 5.1.7, the isomorphism problem for automatic strongly locally finite graphs is Π^0_1-complete. Furthermore, there exists a computable isomorphism between any two automatic copies of a strongly locally finite graph. We present results towards characterizing strongly locally finite graphs that are computably categorical. Firstly, we present a necessary and sufficient condition for certain classes of strongly locally finite graphs to be computably categorical. Then we prove that if there exists an infinite Δ^0_2-set of components that can be properly embedded into infinitely many components of the graph then the graph is not computably categorical. Finally, we construct a strongly locally finite computably categorical graph with a infinite chain of properly embedded components.

6.1 Computable categoricity of graphs

Recall from Defintion 2.5.10 that a structure is *computable* if its domain is a computable subset of natural numbers and its atomic relations are uniformly computable. If $\mathcal{S}$ is a computable structure isomorphic to a structure $\mathcal{S}'$ then $\mathcal{S}$ is called a *computable presentation of* $\mathcal{S}'$ and $\mathcal{G}$ is called *computably presentable*.

Definition 6.1.1 *Two computable structures $\mathcal{S}_1$ and $\mathcal{S}_2$ have the same* computable isomorphism type *if there is a computable isomorphism from $\mathcal{S}_1$ to $\mathcal{S}_2$. The number of computable isomorphism types of the graph $\mathcal{S}$, denoted by* $\dim(\mathcal{S})$*, is called the* computable dimension of $\mathcal{S}$*. If the computable dimension of $\mathcal{S}$ equals 1 then the graph $\mathcal{G}$ is called* computably categorical*.*

Example 6.1.2 *The graph* $(\mathbb{N}; E)$ *where* $E = \{(i, i+1) \mid i \in \mathbb{N}\}$ *is computably categorical. Indeed, given two computable presentations* $\mathcal{G}_1, \mathcal{G}_2$ *of* $(\mathbb{N}; E)$*, a computable isomorphism can be built by first non-uniformly mapping the unique elements with degree 1 in both presentations and then mapping the successor elements in both presentations in parallel.*

By convention, a structure has computable dimension ω if its computable dimension is countably infinite.

Example 6.1.3 *The graph consisting of* $\aleph_0$ *many copies of* $(\mathbb{N}; E)$ *is not computably categorical; in fact, it has computable dimension* ω*.*

In general, providing examples of computably categorical structures or structures of computable dimension ω is easy. S. S. Goncharov in [40] was the first to provide examples of graphs of computable dimension n, where $n > 1$.

The study of computably categorical structures constitutes one of the major topics in the study of computable isomorphisms. Here the goal is to provide a characterization of computably categorical structures within specific classes of structures. This has been done for Boolean algebras [18], linearly ordered sets [99], trees [84], Abelian groups [34], ordered Abelian groups [36], etc.

In this chapter we study computable isomorphisms for a specific class of graphs. We assume all graphs are undirected. Hence an edge with end nodes u and v is written as $\{u, v\}$, which represents the pair of edges (u, v) and (v, u). Since all nodes are natural numbers, we can compare two nodes by saying that a node u is smaller or greater than another node v.

Let $\mathcal{G} = (V; E)$ be a graph. Recall that a component is a maximal subset of V in which any two nodes are connected by a path. Let S be a sequence $\mathcal{G}_0, \mathcal{G}_1, \ldots$ of pairwise disjoint finite graphs. Define the new graph $\mathcal{G}_S$ as the disjoint union of these graphs. More formally, the set of nodes of $\mathcal{G}_S$ is $\bigcup_{i \in \mathbb{N}} V_i$ and the set of edges is $\bigcup_{i \in \mathbb{N}} E_i$. We recall the following definition from Chapter 5.

Definition 5.1.6 A graph $\mathcal{G}$ is *strongly locally finite* if every component of $\mathcal{G}$ forms a finite graph.

It is not hard to see that $\mathcal{G}$ is strongly locally finite if and only if $\mathcal{G} \cong \mathcal{G}_S$ for some sequence S of pairwise disjoint finite graphs. For notational simplicity, we will sometimes identify a set X with its characteristic string $w_X \in \{0,1\}^\omega$, i.e., for all $n \in \mathbb{N}$, $w_X(n) = 1$ iff $n \in X$. We write $X(n)$ for $w_X(n)$. The following proposition gives a full description of computable dimensions for strongly locally finite graphs:

Proposition 6.1.4 *The computable dimension of any strongly locally finite graph is either 1 or* ω*. In particular, no strongly locally finite graph has a finite computable dimension* n*, where* $n > 1$*.*

To prove Prop. 6.1.4, we invoke a well-known result of Goncharov [41].

Theorem 6.1.5 (Goncharov) *If any two computable presentations of a structure $\mathcal{A}$ are isomorphic via a Δ^0_2-function then the computable dimension of $\mathcal{A}$ is either 1 or ω.*

Let $\mathcal{G}$ be a strongly locally finite graph. Recall that a set $X \subseteq \mathbb{N}$ belongs to the class Δ^0_2 if and only if X is computable in $0'$ (the halting problem). By the limit lemma (see [109, p. 56]), a set $X \subseteq \mathbb{N}$ belongs to the class Δ^0_2 if and only if there is a computable sequence[1] $X_0, X_1, \cdots$ of finite subsets of $\mathbb{N}$ such that for all $n \in \mathbb{N}$, the sequence $X_0(n)X_1(n)\cdots$ converges to the value $X(n)$, i.e., $X_i(n) \neq X(n)$ for only finitely many $i \in \mathbb{N}$. Such a sequence $X_0, X_1, \cdots$ is called a Δ^0_2-*approximation* of X.

Proof of Prop. 6.1.4 We show that between any two computable presentations of $\mathcal{G}$ there is a Δ^0_2-isomorphism. Indeed, fix two computable presentations $\mathcal{G}_1 = (\mathbb{N}; E_1), \mathcal{G}_2 = (\mathbb{N}; E_2)$ of $\mathcal{G}$. We informally describe a procedure to set up a Δ^0_2-isomorphism f from $\mathcal{G}_1$ to $\mathcal{G}_2$. The procedure starts by computably enumerating edges in E_1 and E_2 to reveal the components in $\mathcal{G}_1$ and $\mathcal{G}_2$. Once the enumeration reveal two components C_1 in $\mathcal{G}_1$ and C_2 in $\mathcal{G}_2$ that are isomorphic, we define the function f such that it maps C_1 isomorphically to C_2. If at some later stages C_1 becomes non-isomorphic to C_2, the procedure undefines the function f on C_1. Since $\mathcal{G}$ is strongly locally finite, the components C_1 and C_2 may change only finitely many times, and thus the function f will be re-defined on C_1 only finitely often. Since $\mathcal{G}_1 \cong \mathcal{G}_2$, the function f eventually converge to an isomorphism from $\mathcal{G}_1$ to $\mathcal{G}_2$. Note that the sequence of f in all stages of the construction forms a Δ^0_2-approximation and hence $f \in \Delta^0_2$. ∎

By Proposition 6.1.4, it makes perfect sense to work towards a characterization of computably categorical strongly locally finite graphs. This is the subject of this chapter. In the following, we use $\mathfrak{G}_{\mathsf{SLF}}$ to denote the set of computable strongly locally finite graphs.

6.2 Examples of strongly locally finite graphs

In this section, we provide several examples of strongly locally finite graphs with different properties. In our examples all the graphs have components of the following types.

Definition 6.2.1 *1. A* cycle *of length $n > 2$ is a graph isomorphic to $(\{1, ..., n\}; E)$, where $E = \{\{1, 2\}, \{2, 3\}, ..., \{n-1, n\}, \{n, 1\}\}$. We denote this graph by $\mathcal{C}_n$.*

2. A sun *of size $n > 2$ is obtained by attaching a new edge to every node of a cycle of length n. Denote this graph by $\mathcal{S}_n$. The nodes not on the cycle are called* rays.

3. A line *of length $n \geq 1$ is a graph isomorphic to $(\{0, ..., n\}; E)$, where $E = \{\{0, 1\}, \{1, 2\}, ..., \{n-1, n\}\}$. Denote this graph by $\mathcal{L}_n$.*

[1] A sequence of finite sets $X_0, X_1, \cdots$ is *computable* if the function $f(i, n) = X_i(n)$ is computable

4. *The graph C'_n is obtained by attaching exactly* 1 *edge to only one node of C_n.*

5. *The graph C''_n is obtained by attaching exactly* 2 *edges to only one node of C_n.*

Proposition 6.2.2 *There is a graph $\mathcal{G}_1 \in \mathfrak{G}_{\mathsf{SLF}}$ that is not computably categorical.*

Proof. Let $\mathcal{G}_1$ be the graph that contains a copy of $\mathcal{L}_n$ for each $n \geq 1$. This graph is not computably categorical. To illustrate this, we construct two computable presentations of $\mathcal{G}_1$ that are not computably isomorphic. Let $\mathcal{H}_1 = (\mathbb{N}; E)$ denote the *standard computable presentation* of $\mathcal{G}_1$, in which the lines appear in order, i.e., the line $\mathcal{L}_n$ is represented by $\{m, m+1, \ldots, m+n-1\}$ (the edges are between successive elements) where $m = \sum_{i=1}^{n-1} i$. We then build a computable presentation $\mathcal{H}_2 \cong \mathcal{G}_1$ by stages as follows:

We construct a sequence of graph $\mathcal{H}_{2,0} \subset \mathcal{H}_{2,1} \subset \cdots$ such that $\mathcal{H}_2 = \bigcup\{\mathcal{H}_{2,s} \mid s \in \mathbb{N}\}$. At each stage s, for each n, $\mathcal{H}_{2,s}$ will contain at most one copy of $\mathcal{L}_n$ as its component. We write $\mathcal{L}^2_{n,s}$ for the copy of $\mathcal{L}_n$ in $\mathcal{H}_{2,s}$ and write $\mathcal{L}^1_n$ for the copy of $\mathcal{L}_n$ in $\mathcal{H}_1$. Initially no Φ_e, $e \in \mathbb{N}$, is *processed*.

Stage s: If $\mathcal{H}_{2,s}$ does not contain a copy of $\mathcal{L}_s$, include one. For any $e < s$ where Φ_e is not processed, if $\Phi_{e,s}(v) \downarrow$ for some $v \in \mathcal{L}^2_{2e,s}$, then proceed as follows. If $\Phi_{e,s}(v) \notin \mathcal{L}^1_{2e}$, then no action is required. If $\Phi_{e,s}(v) \in \mathcal{L}^1_{2e}$, then extend the copy of $\mathcal{L}^2_{2e,s}$ to a copy of $\mathcal{L}_n$, where n is the least odd number greater than $2e$ such that $\mathcal{L}_n$ does not appear in $\mathcal{H}_{2,s}$. Insert a new copy of $\mathcal{L}_{2e}$ into $\mathcal{H}_{2,s+1}$ and declare Φ_e processed.

Since each component of $\mathcal{H}_2$ is extended and re-introduced at most once after it was first introduced, we certainly build a structure isomorphic to $\mathcal{G}_1$. For each $e \in \mathbb{N}$, if Φ_e is total then there exists a stage $s > e$ where $\Phi_{e,s}(v) \downarrow$ for some $v \in \mathcal{L}^2_{2e,s}$, and at that stage we ensure that Φ_e is not an isomorphism between $\mathcal{H}_1$ and $\mathcal{H}_2$. ∎

Let $\mathcal{G} \in \mathfrak{G}_{\mathsf{SLF}}$ be a computable graph with node set V. The component containing a node v is denoted by $C(v)$.

Definition 6.2.3 *We define the* size function *as* $\mathrm{size}_{\mathcal{G}} : V \to \mathbb{N}$ *such that* $\mathrm{size}_{\mathcal{G}}(v) = |C(v)|$.

Note that the size function $\mathrm{size}_{\mathcal{H}_1}$ of the standard computable presentation $\mathcal{H}_1$ of $\mathcal{G}_1$ is computable.

Proposition 6.2.4 *There is a computably categorical graph $\mathcal{G}_2 \in \mathfrak{G}_{\mathsf{SLF}}$ such that in all computable presentations of the graph the size function is not computable.*

Proof. The desired graph $\mathcal{G}_2$ is obtained as follows. Recall from Chapter 2.3 that $K = \{e \mid \Phi_e(e) \downarrow\}$ is computably enumerable but not computable. Let $\mathcal{G}_2$ be the graph that has, for $n > 2$, a copy of C_n if $n \notin K$ and a copy of $\mathcal{S}_n$ if $n \in K$. A computable presentation of $\mathcal{G}_2$ can be built by simultaneously building the graph consisting of C_n for all $n > 2$, and

enumerating the set K. When a number n is enumerated in K, if a copy of C_n is already created in $\mathcal{G}_2$, then extend it to a copy of $\mathcal{S}_n$; otherwise, create a copy of $\mathcal{S}_n$.

Suppose $\mathcal{H}_1, \mathcal{H}_2$ are two computable presentations of $\mathcal{G}_2$. For a component $C \in \{C_n \mid n \in \mathbb{N}\} \cup \{\mathcal{S}_n \mid n \in \mathbb{N}\}$, we use C^i to denote the copy of C in $\mathcal{H}_i$, $i \in \{1, 2\}$. We build a computable isomorphism between $\mathcal{H}_1$ and $\mathcal{H}_2$ as follows:

For any node v in $\mathcal{H}_1$, search for all nodes in the component $C(v)$ of v until all nodes in the cycle of $C(v)$ have been found. At this point, if some rays in $C(v)$ have already been detected, then $C(v) \cong \mathcal{S}_n$ for some $n \in \mathbb{N}$. In this case, search to find $\mathcal{S}_n^2$ in $\mathcal{H}_2$ and map $\mathcal{S}_n^1$ isomorphically to $\mathcal{S}_n^2$. On the other hand, if no ray in $C(v)$ has been detected, search to find a cycle of length n in $\mathcal{H}_2$, and map these two copies isomorphically. If it turns out that $C(v) \cong \mathcal{S}_n$ for some $n \in \mathbb{N}$, then rays in $C(v)$ will be eventually found, and the map can be extended to an isomorphism from $C(v)$ to $\mathcal{S}_n^2$.

Given any computable presentation $\mathcal{H}$ of $\mathcal{G}_2$ for which the size function is computable, we can use $\text{size}_{\mathcal{H}}$ to compute K. Indeed, for any $n \in \mathbb{N}$, we can search to find a cycle of length n and compute $\text{size}_{\mathcal{H}}(v)$ for a node v on this cycle. Then for $n > 2$, $n \in K$ if and only if $\text{size}_{\mathcal{H}}(v) = 2n$. Hence the size function is not computable in any computable presentations of $\mathcal{G}_2$. ∎

Proposition 6.2.5 *There is a graph $\mathcal{G}_3 \in \mathfrak{G}_{\mathsf{SLF}}$ that is not computably categorical and the size function is not computable in any computable presentation of $\mathcal{G}_3$.*

Proof. Let $\mathcal{G}_3$ be the disjoint union of the graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ described above. Then $\mathcal{G}_3$ is not computably categorical for the same reason that $\mathcal{G}_1$ is not, and the size function on $\mathcal{G}_3$ is non-computable in any computable presentation for the same reason as on $\mathcal{G}_2$. ∎

Given two finite graphs $\mathcal{H}_1 = (V_1; E_1)$ and $\mathcal{H}_2 = (V_2; E_2)$, we say $\mathcal{H}_1$ *properly embeds* into $\mathcal{H}_2$ if V_1 can be mapped injectively to a *proper* subset of V_2 that preserves the edge relation. In this case, we write $\mathcal{H}_1 \prec \mathcal{H}_2$.

Definition 6.2.6 *For a graph $\mathcal{G}$, define the* proper extension function of $\mathcal{G}$ *as* $\text{ext}_{\mathcal{G}} : V \to \mathbb{N} \cup \{\infty\}$ *such that* $\text{ext}_{\mathcal{G}}(v) = |\{C(x) \mid C(v) \prec C(x), x \in \mathbb{N}\}|$. *That is,* $\text{ext}_{\mathcal{G}}(v)$ *is the number of components in which $C(v)$ can be properly embedded.*

We now consider a graph $\mathcal{G} \in \mathfrak{G}_{\mathsf{SLF}}$ where $\text{ext}_{\mathcal{G}}$ is computable and the range of the proper extension function $\text{Rng}(\text{ext}_{\mathcal{G}})$ is a subset of $\mathbb{N}$. The next example shows that this condition does not guarantee computable categoricity of $\mathcal{G}$.

Proposition 6.2.7 *There exists a graph $\mathcal{G}_4 \in \mathfrak{G}_{\mathsf{SLF}}$ that is not computably categorical and* $\text{ext}_{\mathcal{G}_4}$ *is computable and* $\text{Rng}(\text{ext}_{\mathcal{G}}) \subseteq \mathbb{N}$.

Proof. We will simultaneously construct two isomorphic computable graphs $\mathcal{H}_1, \mathcal{H}_2$ that are not computably isomorphic. This shows that the graph is not computably categorical. The construction proceeds by stages. At stage s, $s \in \mathbb{N}$, we construct graphs $\mathcal{H}_{1,s}$ and $\mathcal{H}_{2,s}$ as follows:

Add copies of C_s and C'_s in both $\mathcal{H}_{1,s}$ and $\mathcal{H}_{2,s}$. Name the copies in $\mathcal{H}_{i,s}$ ($i \in \{1,2\}$) C^i_s, C'^i_s respectively. If $\exists e \leq s : \Phi_{e,s}(v) \downarrow \in C^2_e$ for some $v \in C^1_e$, then let e be the least such e, extend C^1_e to a copy of C'_e and extend C'^1_e to a copy of C''_e. In the other copy, extend C^2_e to a copy of C''_e. This ensures that Φ_e is not an isomorphism, but maintains $\mathcal{H}_{1,s+1} \cong \mathcal{H}_{2,s+1}$. Let $\mathcal{H}_i = \bigcup\{\mathcal{H}_{i,s} \mid s \in \mathbb{N}\}$, $i \in \{1,2\}$. It is easy to see that $\mathcal{H}_1 \cong \mathcal{H}_2$ but no Φ_e computes an isomorphism.

Moreover, for each node $v \in \mathbb{N}$, $\mathrm{ext}_{\mathcal{H}_1}(v) \in \{0,1\}$. To compute $\mathrm{ext}_{\mathcal{H}_1}(v)$, run the construction until the stage where v is added to $\mathcal{H}_1$. If v is put in a copy of (or to extend a copy of) C_e, then $\mathrm{ext}_{\mathcal{H}_1}(v) = 1$, if v is put in a copy of (or to extend a copy of) C'_e, then $\mathrm{ext}_{\mathcal{H}_1}(v) = 0$. Hence $\mathrm{ext}_{\mathcal{H}_1}$ is a computable function. The desired graph $\mathcal{G}_4$ is $\mathcal{H}_1$. ∎

Our final example is of a structure that is computably categorical and yet whose proper extension function is not computable. This, together with the previous example, shows that there is no good relation between computability of the proper extension function and computable categoricity of the graph.

Proposition 6.2.8 *There is a computably categorical graph $\mathcal{G}_5 \in \mathfrak{G}_{\mathsf{SLF}}$ on which the proper extension function is non-computable.*

Proof. Let $\mathcal{G}_5$ be the graph that has, for all $n > 2$, one copy of C_n and one copy of $\mathcal{S}_n$ if $n \notin K$, and two copies of $\mathcal{S}_n$ if $n \in K$. This structure is clearly computably presentable in the same reason as $\mathcal{G}_2$ above. It is also computably categorical. Indeed, to define a computable isomorphism between two computable presentations, first for each n match up the first copies of $\mathcal{S}_n$ that are found. Then, match up the copies of C_n. If the copy of C_n ends up being extended to S_n, then this must happen in both copies, and the isomorphism can be extended.

Given $\mathrm{ext}_{\mathcal{H}}$ on any computable presentation $\mathcal{H}$ of $\mathcal{G}_5$, one can compute K. Indeed, find two distinct copies of C_n in $\mathcal{G}_5$, and let v_1 be a node from one copy, and v_2 a node from the other copy. Then $n \in K$ if and only if $\mathrm{ext}_{\mathcal{H}}(v_1) + \mathrm{ext}_{\mathcal{H}}(v_2) = 0$. ∎

6.3 Graphs with computable size functions

In this section we continue to investigate the relationship between computable categoricity and the size function of a strongly locally finite graph.

Lemma 6.3.1 *Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be computable presentations of $\mathcal{G} \in \mathfrak{G}_{\mathsf{SLF}}$ such that $\mathrm{size}_{\mathcal{G}_1}, \mathrm{size}_{\mathcal{G}_2}$ are computable. Then $\mathcal{G}_1$ and $\mathcal{G}_2$ are computably isomorphic.*

Proof. For $i \in \{1, 2\}$ and any node v in the graph $\mathcal{G}_i$, we can effectively reveal the component $C(v)$ by finding all $\text{size}_{\mathcal{G}_i}(v)$ nodes that are connected to v by a path. We use a standard back-and-force argument to set up a computable isomorphism f between $\mathcal{G}_1$ and $\mathcal{G}_2$ as follows: Pick a node v in $\mathcal{G}_1$, compute its component $C(v)$, and find in $\mathcal{G}_2$ a component $C' \cong C(v)$ such that f has not mapped any element in $\mathcal{G}_1$ to C'. We then let f map $C(v)$ isomorphically to C'. For the other direction, pick a node $v' \in \mathcal{G}_2$ and repeat the above procedure by replacing f with f^{-1} and $\mathcal{G}_1$ with $\mathcal{G}_2$. ∎

Let $\mathcal{G} \in \mathfrak{G}_{\mathsf{SLF}}$. The lemma implies that $\mathcal{G}$ is computably categorical if the size function is *intrinsically computable*, that is, if it is computable for all computable presentation of $\mathcal{G}$.

Proposition 6.3.2 *Suppose* $\text{size}_{\mathcal{G}}$ *is a computable function. The graph* $\mathcal{G}$ *is computably categorical if and only if the size function is intrinsically computable.*

Proof. One direction is proved by Lemma 6.3.1. The other direction is straightforward. Suppose $\mathcal{G}$ is computably categorical. Then from any of its computable presentation $\mathcal{G}'$ to $\mathcal{G}$ there is a computable isomorphism h. Then $\text{size}_{\mathcal{G}'}(v) = \text{size}_{\mathcal{G}}(h(v))$ for any v in $\mathcal{G}'$. ∎

We now further exploit the case when the size function $\text{size}_{\mathcal{G}}$ is computable. As illustrated in the proof of Lemma 6.3.1, when $\text{size}_{\mathcal{G}}$ is computable, one may effectively reveal the component $C(v)$ of any node v using the function $\text{size}_{\mathcal{G}}(v)$. In this way, we have an effective list (without repetition) $C_0, C_1, \ldots$ of all components in $\mathcal{G}$. For $s \in \mathbb{N}$, let $\mathcal{G}_s$ be the graph $\mathcal{G}$ restricted on the components $C_0, \ldots C_{s-1}$. The next lemma gives a necessary condition for a graph $\mathcal{G} \in \mathfrak{G}_{\mathsf{SLF}}$ to be computably categorical.

Lemma 6.3.3 *Suppose* $\text{size}_{\mathcal{G}}$ *is computable. If there are infinitely many nodes* v *such that* $ext_{\mathcal{G}}(v) = \infty$*, then* $\mathcal{G}$ *is not computably categorical.*

Proof. Our goal is to build a graph $\mathcal{G}' = (\mathbb{N}; E')$ such that $\mathcal{G}' \cong \mathcal{G}$ but $\mathcal{G}'$ is not computably isomorphic to $\mathcal{G}$. We construct an infinite ascending chain of finite graphs $\mathcal{G}'_0 \subset \mathcal{G}'_1 \subset \ldots$ whose limit is our desired $\mathcal{G}'$, i.e., $\mathcal{G}' = \bigcup_s \mathcal{G}'_s$. At stage s we construct $\mathcal{G}'_s$ and f_s such that $\mathcal{G}'_s$ is isomorphic to $\mathcal{G}_s$, where f_s is the isomorphism. Recall from Chapter 2.3 that $\Phi_0, \Phi_1, \ldots$ is a standard enumeration of all partial computable functions from $\mathbb{N}$ to $\mathbb{N}$. For each e we meet the following requirement:

$$P_e : \Phi_e \text{ is not an isomorphism from } \mathcal{G} \text{ to } \mathcal{G}'$$

Initially all components in $\mathcal{G}'$ are *free* for Φ_e where $e \in \mathbb{N}$, and no Φ_e is *processed*.

Construction. At stage 0, set $\mathcal{G}'_0$ as the empty graph and f_0 is undefined on any $x \in \mathbb{N}$. At stage $s + 1$, consider $\mathcal{G}_{s+1}$ obtained by adding C_s to $\mathcal{G}_s$. Let $C'_0, ..., C'_{s-1}$ be the components in $\mathcal{G}'_s$ such that f_s maps C'_i, $i < s$, isomorphically to C_i. Find the least $e \leq s + 1$ such that for some $i < s$ we have:

- $C_i \prec C_s$.
- Φ_e has not been processed and Φ_e is defined on C_i.
- $\Phi_{e,s+1}$ is a partial isomorphism on $\mathcal{G}_{s+1}$.
- The component $\Phi_e(C_i)$ is free for Φ_e.

If such e does not exist then construct $\mathcal{G}'_{s+1}$ by adding a copy of C_s to $\mathcal{G}'_s$ and extend f_s to f_{s+1} by mapping C'_s isomorphically to C_s. Otherwise, let C'_j be $\Phi_e(C_i)$ and perform the following steps:

1. Extend C'_j to a component, denoted by C'_s, such that $C'_s \cong C_s$.
2. Add a new copy of C_j to $\mathcal{G}'_{s+1}$, denoted by C'_j.
3. Define f_{s+1} by setting $f_{s+1}(x) = f_s(x)$ on all nodes x that do not belong to the components C'_j and C'_s, mapping C'_j isomorphically to C_j and mapping C'_s isomorphically to C_s.
4. Declare C'_s *not* free for all $\Phi_{e'}$ with $e' > e$.
5. Declare Φ_e processed.

This completes the construction for $\mathcal{G}'_{s+1}$ and the function f_{s+1}. It is clear that f_{s+1} is an isomorphism from $\mathcal{G}_{s+1}$ to $\mathcal{G}'_{s+1}$.

Verification. Each requirements P_e is satisfied. Otherwise, let e be the smallest number such that P_e is not satisfied. Let s be the stage when all requirements with smaller indices are satisfied. Note that in $\mathcal{G}'_s$ there are at most e components that are not free for Φ_e. Take $i > s$ such that $\{j \mid C_i \prec C_j\}$ is infinite and $\Phi_e(C_i)$ is free for Φ_e. Let $t > s$ be the first stage where $C_i \prec C_t$ and $\Phi_{e,t}$ is defined on C_i. At stage t, the construction will process Φ_e and ensure Φ_e is not an isomorphism. It is not hard to see that $f(v) = \lim_s f_s(v)$ establishes an isomorphism between $\mathcal{G}'$ and $\mathcal{G}$. ∎

In Prop. 6.2.7 and Prop. 6.2.8 we showed that the proper extension function is unrelated to computable categoricity of graphs in $\mathfrak{G}_{\mathsf{SLF}}$ in general. In the following we show that this is not the case when $\text{size}_{\mathcal{G}}$ is computable.

Lemma 6.3.4 *Suppose* $\text{size}_{\mathcal{G}}$ *is computable and there are only finitely many* v *such that* $ext_{\mathcal{G}}(v) = \infty$. *If* $\text{ext}_{\mathcal{G}}$ *is not computable then* $\mathcal{G}$ *is not computably categorical.*

Proof. The construction of $\mathcal{G}'$ that is isomorphic but not computably isomorphic to $\mathcal{G}$ is very similar to the construction for Lemma 6.3.3. The only difference is that we start with a different $\mathcal{G}_0$ which contains all (finitely many) components in $\mathcal{G}$ that embed into infinitely

many components. Therefore in this construction let $C_0, C_1, \ldots$ list all *other* components in $\mathcal{G}$. The construction of the previous lemma is then repeated.

Suppose e is the smallest number such that P_e is not satisfied. Let s be the stage when all requirements with smaller indices are satisfied. Since Φ_e is an isomorphism, we can compute the function $\text{ext}_{\mathcal{G}}$ as follows. Consider C_i such that $\Phi_e(C_i)$ is free for Φ_e at stage s. Note that there are only finitely many C_i's that are not free for Φ_e. Let $t > s$ be the least stage such that $\Phi_{e,t}$ is defined on C_i. Note that by construction C_i can not be properly embedded into C_k for all $k > t$ as otherwise the procedure will act to satisfy P_e. Hence the number of proper extensions of C_i in $\mathcal{G}$ is the same as the number of proper extensions of C_i in $\mathcal{G}_t$, which can be computed effectively. ∎

We can now prove the following characterization theorem.

Theorem 6.3.5 *Let $\mathcal{G} \in \mathfrak{G}_{\mathsf{SLF}}$ be a graph such that $\text{size}_{\mathcal{G}}$ is a computable function. Then the following are equivalent:*

(1) $\mathcal{G}$ is computably categorical.

(2) The size function is intrinsically computable.

(3) There are finitely many v such that $\text{ext}_{\mathcal{G}}(v) = \infty$ and the function $\text{ext}_{\mathcal{G}}$ is computable.

Proof. The equivalence of (1) and (2) follows from Proposition 6.3.2. The direction (1) to (3) follows from the lemmas above. We prove the implication (3) to (1).

Let $\mathcal{G}'$ be a computable presentation of $\mathcal{G}$. Take all (finitely many) components C_i such that $\{j \mid C_i \prec C_j\}$ is infinite. Non-uniformly map them to isomorphic components in $\mathcal{G}'$.

Take C_i such that $\{j \mid C_i \prec C_j\}$ is finite. By using $\text{ext}_{\mathcal{G}}(v)$ for some $v \in C_i$, find components $X, X_1, \ldots X_{\text{ext}_{\mathcal{G}}(v)}$ in $\mathcal{G}'$ such that $X \cong C_i$ and for each $\ell \in \{1, ..., \text{ext}_{\mathcal{G}}(i)\}$, $C_i \prec X_\ell$. Map C_i isomorphically to X. There are no more components in $\mathcal{G}'$ that properly extends C_i. Hence X is a component in $\mathcal{G}'$ isomorphic to C_i. ∎

6.4 A sufficient condition for non-computably categoricity

In this section we do not assume computability of the size function $\text{size}_{\mathcal{G}}$ and hence we do not have an effective list $C_0, C_1, \ldots$ of all components in $\mathcal{G}$. For this case, we prove the following theorem which provides a sufficient condition for a graph to be not computably categorical.

Theorem 6.4.1 *Suppose $\mathcal{G} \in \mathfrak{G}_{\mathsf{SLF}}$. If there exists an infinite Δ^0_2 set of nodes X in $\mathcal{G}$ such that $ext_{\mathcal{G}}(x) = \infty$ for all $x \in X$, then $\mathcal{G}$ is not computably categorical.*

Proof. Let $\mathcal{G} = (G; E) \in \mathfrak{G}_{\mathsf{SLF}}$ and X be an infinite Δ^0_2 set of nodes such that $\mathrm{ext}_{\mathcal{G}}(x) = \infty$ for all $x \in X$. We build a computable presentation $\mathcal{G}' = (\mathbb{N}; E')$ of $\mathcal{G}$ that meets the following requirement for each $e \in \mathbb{N}$:

$$P_e : \Phi_e \text{ is not an isomorphism from } \mathcal{G} \text{ to } \mathcal{G}'$$

We will define a chain $G_0 \subset G_1 \subset G_2 \subset \ldots$ of subsets of $\mathbb{N}$ such that each $G_s \supseteq \{0, 1, \ldots, s\}$. Note that $\lim_s G_s = \mathbb{N}$ and for $s \in \mathbb{N}$ the graph $\mathcal{G}_s = (G_s; E \upharpoonright G_s)$ is a subgraph of $\mathcal{G}$. We will construct an infinite chain of finite graphs $\mathcal{G}'_0 \subset \mathcal{G}'_1 \subset \ldots$ whose limit is our desired $\mathcal{G}'$. At stage s we construct $\mathcal{G}'_s$ and f_s such that $\mathcal{G}'_s$ is isomorphic to $\mathcal{G}_s$, where f_s is the isomorphism. For $s \in \mathbb{N}$ and a node $v \in \mathcal{G}_s \cup \mathcal{G}'_s$, we use $C_s(v)$ to denote the component containing v in $\mathcal{G}'_s \cup \mathcal{G}_s$.

Let $X_0, X_1, \ldots$ be a Δ^0_2-approximation of X. Each X_s induces a finite subgraph $\mathcal{X}_s$ of $\mathcal{G}$. For $v \in G_s$, we may assume without loss of generality that $v \in X_s$ implies $u \in X_s$ for all $u \in C_s(v)$. Let $x_{0,s}, x_{1,s}, \ldots$ denote the least nodes (in the natural number ordering) in all components in $\mathcal{X}_s$ in increasing order. Since $\mathcal{G}$ is strongly locally finite and $X \in \Delta^0_2$, the number $x_n = \lim_s x_{n,s}$ exists for all $n \in \mathbb{N}$.

For each e where Φ_e is defined on all $\mathbb{N}$, we will pick a node v_e in $\mathcal{G}$ that serves as a *witness* for Φ_e not being an isomorphism from $\mathcal{G}$ to $\mathcal{G}'$, i.e., $\Phi_e(C(v_e)) \not\cong C(v_e)$. At each stage s where $s > e$, we set $v_{e,s} = x_{n,s}$ for some n. We will need to make sure that $v_e = \lim_s v_{e,s}$ exists.

We first describe the construction for meeting a single requirement P_0. The general construction (for multiple requirements) will be described later. Let s be the first stage where $\Phi_{0,s}$ is defined on all nodes in $C_s(x_{0,s})$ and is a partial isomorphism from $\mathcal{G}_s$ to $\mathcal{G}'_s$. Note that at this stage, an isomorphism f_s from $\mathcal{G}_s$ to $\mathcal{G}'_s$ is also defined. We let $v_{0,s} = x_{0,s}$ and start two processes simultaneously:

(a) Enumerate nodes in $\mathcal{G}$ to look for a component C in $\mathcal{G}$ that is disjoint from the range of f_s and $C_s(v_{0,s}) \prec C$.

(b) Compute the nodes $x_{0,s}, x_{0,s+1}, x_{0,s+2}, \ldots$.

There are two possibilities:

– Case 1. The process (a) returns with a component C as specified. In this case, we do the following:

1. Add the component C to $\mathcal{G}_s$.
2. Add a new component D to $\mathcal{G}'_s$ that is isomorphic to $f_s^{-1}(C_s(\Phi_0(x_{0,s}))) \cong C_s(x_{0,s})$.
3. Extend the component $C_s(\Phi_0(x_{0,s}))$ in $\mathcal{G}'_s$ so that it is isomorphic to C.

4. Re-define f_s by mapping $f_s^{-1}(C_s(\Phi_0(x_{0,s})))$ isomorphically to D, and mapping C isomorphically to the extended component $C_s(\Phi_0(x_{0,s}))$. In this way, f_s remains an isomorphism from $\mathcal{G}_s$ to $\mathcal{G}'_s$.

– Case 2. If $x_{0,s} \notin X$, the desired component C would never be found. In this case, there is $s' > s$ such that $x_{0,s'} \neq x_{0,s}$. We will notice this and continue to another stage of the construction.

Note that if Case 1 is reached at stage s, then $C_s(v_{0,s}) \prec C_s(\Phi_0(v_{0,s}))$. The function Φ_e is not an isomorphism from $\mathcal{G}$ to $\mathcal{G}'$ until the component $C(v_{0,s})$ is extended and become isomorphic to the component $C(\Phi_0(v_{0,s}))$. When this occurs, we perform the above operations again to look for another component C. Since $\mathcal{G}$ is strongly locally finite, the component $C(v_{0,s})$ can be extended for at most finitely many times.

Suppose P_0 is not satisfied. Let s be the least stage such that $x_{0,s'} = x_0$ for all $s' \geq s$ and the component of x_0 has fully reveals itself, i.e., $C_s(x_0) = C(x_0)$. Since Φ_0 is an isomorphism from $\mathcal{G}$ to $\mathcal{G}'$, there is a stage $s' \geq s$ where $\Phi_{0,s'}$ is defined on all nodes in $C(x_0)$ and is a partial isomorphism. At stage s' we run the above procedure on x_0 and since $x_0 \in X$, we will eventually find a component C disjoint from the range of $f_{s'}$ such that $C_{s'}(x_{0,s'}) \prec C$. We then run Case 1 above. Therefore, x_0 is a true witness for satisfying the requirement P_0, i.e., $C(x_0) \prec C(\Phi_0(x_0))$, which guarantees that Φ_0 is not an isomorphism from $\mathcal{G}$ to $\mathcal{G}'$. This is a contradiction with our assumption.

We now describe the construction for the case of multiple requirements. The only extra complication for multiple requirements is that we want to ensure that f is an isomorphism from $\mathcal{G}$ to $\mathcal{G}'$. So for all $v \in \mathbb{N}$, we need to make sure $f(v)$ exists, i.e., we only re-define $f(v)$ for finitely many times. We say that a requirement P_e has *higher priority* than P_t if $e < t$. If we find that $\Phi_e(v_{e,s}) \downarrow$, but is mapped to some component where we have already re-defined f for the sake of higher priority requirements, then instead of proceeding with the diagonalization, we will change v_e to be the next member of X (i.e., if $v_{e,s} = x_{n,s}$, we would let $v_{e,s+1} = x_{n+1,s+1}$). Since each requirement only causes f to be re-defined finitely often and there are only finitely many requirements with higher priorities than P_e, v_e will be re-defined finitely often for this reason. We now give the formal construction.

Construction. At stage 0, let $G_0 = \{0\}$ and $\mathcal{G}'_0$ be the graph consisting of only one node, which is mapped from 0 by the function f_0. At stage $s+1$, $s \geq 0$, suppose we have defined $\mathcal{G}_s, \mathcal{G}'_s$ and an isomorphism $f_s : \mathcal{G}_s \to \mathcal{G}'_s$. Enumerate a new element into $\mathcal{G}_s$ and extend $\mathcal{G}'_s$ correspondingly to preserve the isomorphism f_s. Compute the nodes $x_{0,s+1}, x_{1,s+1}, \ldots, x_{k,s+1}$ where k is the number of components in the finite graph X_{s+1}. Then perform the following steps:

Step 1. Choose the least e such that $\Phi_{e,s+1}(v_{e,s}) \downarrow$ and $C_{s+1}(v_{e,s}) \cong C_{s+1}(\Phi_{e,s+1}(v_{e,s}))$, and such that $x_{n,s+1} = x_{n,s}$, where n is such that $x_{n,s} = v_{e,s}$. If no such e exists, move to Step 2. If f^{-1} or f

have already been re-defined at earlier stages by higher priority requirements on $\Phi_{e,s+1}(v_{e,s})$ or $f^{-1}(\Phi_{e,s+1}(v_{e,s}))$, respectively, then set $v_{e,s+1} = x_{n+1,s+1}$. For $i > e$, let $v_{i,s+1} = x_{n+1+i-e,s+1}$. For $i < e$, let $v_{i,s+1} = x_{m,s+1}$, where m is such that $x_{m,s} = v_{i,s}$. Move to stage $s + 2$.

Otherwise, speed up the enumeration of $\mathcal{G}$ and the approximation of X until we find some $t > s$ where one of the two cases below occurs:

Case 1. There is $v \in \mathcal{G}_t$ such that f_s is not defined on v and $C_t(v_{e,s+1}) \prec C_t(v)$.

Case 2. $x_{n,t} \neq x_{n,s}$ where $v_{e,s} = x_{n,s}$.

In Case 1, move to Step 2. Otherwise, move to Step 3.

Step 2. Perform the following operations:

1. Add the component $C_t(v)$ to $\mathcal{G}_{s+1}$.
2. Add a new component D to $\mathcal{G}'_{s+1}$ that is isomorphic to $C_{s+1}(f_s^{-1}(\Phi_e(v_{e,s})))$.
3. Extend the component $C_{s+1}(\Phi_e(v_{e,s}))$ in $\mathcal{G}'_s$ so that it is isomorphic to $C_t(v)$.
4. Re-define f_s by mapping $C_{s+1}(f_s^{-1}(\Phi_e(v_{e,s})))$ isomorphically to D, and mapping $C_t(v)$ isomorphically to the extended component $C_{s+1}(\Phi_e(v_{e,s}))$. In this way, f_{s+1} remains an isomorphism from $\mathcal{G}_{s+1}$ to $\mathcal{G}'_{s+1}$.

This finishes the construction at this stage for Case 1.

Step 3. Let n be least such that $x_{n,s+1} \neq x_{n,s}$. For e such that $v_{e,s} = x_{m,s}$ with $m < n$, let $v_{e,s+1} = v_{e,s}$. Let e be least such that $v_{e,s} = x_{m,s}$ with $m \geq n$. For $i \in \{e, \ldots, s\}$, let $v_{i,s+1} = x_{n+i-e,s+1}$. This finishes the construction at this stage for Case 2.

Verification. The correctness of the construction is based on the following two claims.

Claim 1. Each requirement P_e, $e \in \mathbb{N}$, is satisfied.

Suppose P_e is the first requirement not satisfied. Note that $v_e = \lim_s v_{e,s}$ exists as v_e will no longer be re-defined after all requirements with higher priorities have been satisfied and v_i have been stablized for all $i < e$. Let s be the least stage such that $v_{e,s'} = v_e$ for all $s' \geq s$ and the component of v_e has fully reveals itself, i.e., $C_s(v_e) = C(v_e)$. Since Φ_e is an isomorphism from $\mathcal{G}$ to $\mathcal{G}'$, there is a stage $t \geq s$ where $\Phi_{e,t}$ is defined on all nodes in $C(v_e)$ and is a partial isomorphism. At stage t, since $x_e \in X$, we will find a node v such that $C_t(v)$ is disjoint from the range of f_t and $C_t(v_e) \prec C_t(v)$. The construction will ensures that v_e is a true witness for satisfying the requirement P_e. Hence Φ_e is not an isomorphism from $\mathcal{G}$ to $\mathcal{G}'$. This is a contradiction with our assumption. This proves Claim 1.

Claim 2. $\mathcal{G} \cong \mathcal{G}'$

By construction, at each stage s, f_s is an isomorphism from $\mathcal{G}_s$ to $\mathcal{G}'_s$. To show that $\mathcal{G} \cong \mathcal{G}'$, it suffices to show that for each $v \in \mathcal{G}$, $f(v) = \lim_s f_s(v)$ exists. If v do not belong to $C(v_{e,s})$ for any e, s, then f would never be re-defined on v. Suppose $v \in C(v_{e,s})$ for some $e, s \in \mathbb{N}$. Let e be the maximum number such that $v \in C(v_{e,s})$ for some s. After all requirements P_i for $i \leq e$ have been satisfied, f is never re-defined on v. Hence Claim 2 is proved. ∎

Finally, we note that since every infinite Σ^0_2 set has an infinite Δ^0_2 subset, in Theorem 6.4.1, we need only assume that X is an infinite Σ^0_2 set.

6.5 Δ^0_3-chain of embedded components

From Theorem 6.3.5 and Theorem 6.4.1 above, one may suggest that the existence of an infinite chain of properly embedded components in a graph may imply that the graph is not computably categorical. One may also suggest that the Δ^0_2-bound in Theorem 6.4.1 could be replaced with a Δ^0_3-bound. The main result of this section is to refute these two suggestions and prove the following:

Theorem 6.5.1 *There is a computably categorical graph $\mathcal{G}$ in $\mathfrak{G}_{\mathsf{SLF}}$ that possesses an infinite chain of properly embedded components. Furthermore, the set $\{v \mid ext_{\mathcal{G}}(v) = \infty\}$ belongs to Δ^0_3.*

In the following we first describe a subclass of graphs that contains the desired graph $\mathcal{G}$ and encode weighted equivalence structures (see Section 6.5.1 for definitions) into these graphs. To prove Theorem 6.5.1, it suffices to construct a weighted equivalence structure that satisfies some requirements. The subsequent sections are devoted to constructing such a structure.

6.5.1 Special cyclic graphs and weighted equivalence structures

Fix a cycle $C_n = (\{1, 2, \ldots, n\}; \{\{1, 2\}, \{2, 3\}, \ldots, \{n-1, n\}, \{n, 1\}\})$. Let v be a node not in C_n. To *attach C_n to v* means to connect v with C_n by adding the edge $\{v, 1\}$.

Definition 6.5.2 *A graph $\mathcal{G}$ is* special cyclic *if each of its components can be obtained from attaching several (possibly infinitely many) cycles to a node v and the lengths of all cycles are greater than 2 and pairwise distinct. The node v is a* root *in $\mathcal{G}$.*

Figure 6.1 illustrates a special cyclic graph. Note that a special cyclic graph may have infinite components and therefore may not be a member of $\mathfrak{G}_{\mathsf{SLF}}$. To prove Theorem 6.5.1, we will construct a computably categorical special cyclic graph that belongs to $\mathfrak{G}_{\mathsf{SLF}}$.

Definition 6.5.3 *A* weighted equivalence structure *$\mathcal{E}$ is of the form $(V; E, (P_n)_{n\in\mathbb{N}})$ where $(V; E)$ is an equivalence structure and $P_0, P_1, \ldots$ are pairwise disjoint subsets of V. We say that an element*

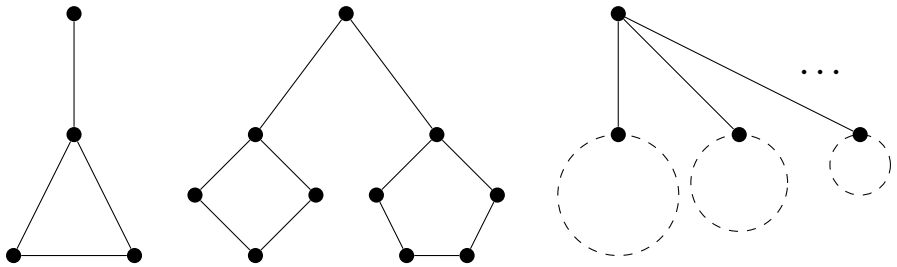


Figure 6.1: A special cyclic graph.

$v \in V$ has weight *n if $v \in P_n$, in this case we denote v by* $\mathbf{n}$*. A weighted equivalence structure $\mathcal{E}$ is* computable *if V, E are computable sets and the mapping from a node to its weight is a computable function*[2]*.*

Let $\mathcal{E}$ be a weighted equivalence structure. We call an E-equivalence class a *component*. Let C be a component in $\mathcal{E}$. We say that C *contains* $\mathbf{n}$, denoted by $\mathbf{n} \in C$, if C contains an element with weight n. A component C embeds in a component D, denoted by $C \preceq D$, if $\mathbf{n} \in C$ implies $\mathbf{n} \in D$ for all $n \in \mathbb{N}$.

Let $\mathcal{E}$ be a computable weighted equivalence structure. We define the computable graph $\mathcal{G}(\mathcal{E})$ as follows: For every element x of $\mathcal{E}$, create a cycle C_x of length $n + 3$ where $x \in P_n$, and put two cycles C_x, C_y in the same special cyclic component whenever $(x, y) \in E$. We use $C(x)$ to denote the component of any node x in $\mathcal{G}(\mathcal{E})$. For a component C in $\mathcal{E}$, we use $\mathcal{G}(C)$ to denote the component in $\mathcal{G}(\mathcal{E})$ that contains all cycles C_x where $x \in C$. A set X of elements in $\mathcal{E}$ is *component-closed* if $x \in X$ implies that all elements in the component of x belong to X.

Proposition 6.5.4 *Let $\mathcal{E}$ be a computable weighted equivalence structure. The following hold:*

(a) For any computable weighted equivalence structure $\mathcal{E}'$, $\mathcal{E} \cong \mathcal{E}'$ if and only if $\mathcal{G}(\mathcal{E}) \cong \mathcal{G}(\mathcal{E}')$.

(b) The graph $\mathcal{G}(\mathcal{E})$ is computably categorical if and only if $\mathcal{E}$ is computably categorical.

(c) For $A \subseteq \mathbb{N}$, and a component-closed set X of elements in $\mathcal{E}$, we have

$$X \leq_{\mathrm{T}} A \Leftrightarrow \{x \mid \exists y \in X : C(x) \text{ contains } C_y\} \leq_{\mathrm{T}} A.$$

Proof. (a) For any computable weighted equivalence structure $\mathcal{E}'$, suppose f is an isomorphism between $\mathcal{E}$ and $\mathcal{E}'$. An isomorphism from $\mathcal{G}(\mathcal{E})$ to $\mathcal{G}(\mathcal{E}')$ can be defined by mapping $\mathcal{G}(C)$ isomorphically to $\mathcal{G}(f(C))$ for all components C of $\mathcal{E}$. The other direction of (a) can be proved in a similar way.

[2] Alternatively, one may require $(P_n)_{n \in \mathbb{N}}$ to be a uniformly computable sequence of sets. These two definition are equivalent.

(b) Suppose $\mathcal{E}$ is computably categorical and let $\mathcal{H} \cong \mathcal{G}(\mathcal{E})$ be computable. From $\mathcal{H}$ we can construct a computable weighted equivalence structure $\mathcal{E}'$ that consists of all nodes in $\mathcal{H}$ that are adjacent to a root, two nodes are equivalent if and only if they are adjacent to the same root, and set the weight of a node u to n if and only if the cycle containing u has length $n+3$. It is easy to see that $\mathcal{G}(\mathcal{E}') \cong \mathcal{H}$. Hence by (a), $\mathcal{E}' \cong \mathcal{E}$. Let f be a computable isomorphism from $\mathcal{E}'$ to $\mathcal{E}$. We can build a computable isomorphism from $\mathcal{H}$ to $\mathcal{G}(\mathcal{E})$ as follows: For any node x that are adjacent to a root in $\mathcal{H}$, map the component containing x isomorphically to the component in $\mathcal{G}(\mathcal{E})$ that contains the cycle $C_{f(x)}$. This proves that $\mathcal{G}(\mathcal{E})$ is computably categorical and one direction of (b). The other direction can be proved in a similar way.

(c) Let X be a component-closed set of elements in $\mathcal{E}$. Suppose $X \leq_T A$. For every node x in $\mathcal{G}(\mathcal{E})$, to decide whether $C(x)$ contains C_y for some $y \in X$, we computably enumerate elements in $\mathcal{E}$ and $\mathcal{G}(\mathcal{E})$ while matching up elements in $\mathcal{E}$ with their corresponding cycles in $\mathcal{G}(\mathcal{E})$. When a cycle that belongs to the same component as x is enumerated, say this cycle is C_z for some z in $\mathcal{E}$, we run the A-oracle computation to check whether $z \in X$. Since X is component-closed, $z \in X$ if and only if $C(x)$ contains C_y for some $y \in X$. For the other direction, let x be an element in $\mathcal{E}$. Then $x \in X$ if and only if some node in C_x belongs to the set $\{z \mid \exists y \in X : C(z) \text{ contains } C_y\}$, which can be checked effectively against an A-oracle computation. ∎

The following proposition is the main technical result of this section.

Proposition 6.5.5 *There is a computable weighted equivalence structure $\mathcal{F}$ such that*

- *each component of $\mathcal{F}$ is finite,*
- *$\mathcal{F}$ is computably categorical, and*
- *$\mathcal{F}$ possesses an infinite chain of properly embedded components.*

Furthermore, the set of elements whose components properly embed into infinitely many components is computable in 0″.

Prop. 6.5.5 suffices for proving Theorem 6.5.1 because by Prop. 6.5.4, the graph $\mathcal{G}(\mathcal{F})$ is computably categorical, all components in $\mathcal{G}(\mathcal{F})$ are finite and $\mathcal{G}(\mathcal{F})$ contains an infinite chain of properly embedded components. Furthermore, by (c) of Prop. 6.5.4, the set $\{v \mid \text{ext}_{\mathcal{G}(\mathcal{F})}(v) = \infty\}$ is computable in 0″ and hence belongs to Δ^0_3. The remaining sections of this chapter is devoted to proving Prop. 6.5.5.

Proposition 6.5.6 *From each $e \in \mathbb{N}$, one uniformly computes a sequence of finite weighted equivalence structures $\mathcal{E}_{e,0} \subset \mathcal{E}_{e,1} \subset \mathcal{E}_{e,2} \subset \ldots$ such that the sequence $\mathcal{E}_0, \mathcal{E}_1, \mathcal{E}_2, \ldots$, where $\mathcal{E}_i = \bigcup_s \mathcal{E}_{i,s}$ for $i \in \mathbb{N}$, lists all computable weighted equivalence structures.*

Proof. We effectively encode pairs of natural numbers $(a, b) \in \mathbb{N}^2$ by a single number $< a, b > \in \mathbb{N}$ (using a standard pairing function such as $< a, b >= (a+b)(a+b+1)/2+b$). For $e \in \mathbb{N}$, define

$$\widehat{\Phi}_{e,t}(x,y) = \begin{cases} 1 & \text{if } \Phi_{e,t}(< x, y >) = 1 \\ 0 & \text{if } \Phi_{e,t}(< x, y >) \downarrow \neq 1 \\ \text{undefined} & \text{if } \Phi_{e,t}(< x, y >) \uparrow \end{cases}$$

and $\widehat{\Phi}_e(x,y) = \lim_s \widehat{\Phi}_{e,s}(x,y)$. Hence, the sequence $\widehat{\Phi}_0, \widehat{\Phi}_1, \ldots$ is a standard enumeration of all partial computable function from $\mathbb{N}^2$ to $\{0,1\}$. Let $\mathsf{Equiv}(i,t,x)$ denote the formula that specifies that the function $\widehat{\Phi}_{i,t}$ converges on all pairs of numbers $y, z \leq x$ and the relation $\{(y,z) \in \{0,\ldots,x\}^2 \mid \widehat{\Phi}_{i,t}(y,z) = 1\}$ is an equivalence relation. For each $i, j, t \in \mathbb{N}$, we define the structure $\mathcal{E}_{<i,j>,t} = (V_{<i,j>,t}; E_{<i,j>,t}, (P_{<i,j>,k,t})_{k\in\mathbb{N}})$ where

$$\begin{aligned} V_{<i,j>,t} &= \{x \mid x < t, \forall y \leq x : \Phi_{i,t}(y) \downarrow \wedge \mathsf{Equiv}(j,t,x)\} \\ E_{<i,j>,t} &= \{(y,z) \in V^2_{<i,j>,t} \mid \widehat{\Phi}_{i,t}(y,z) = 1\} \\ P_{<i,j>,k,t} &= \{x \in V_{<i,j>,t} \mid \Phi_{j,t}(y) = k\} \end{aligned}$$

Note that $V_{<i,j>,t} \subseteq V_{<i,j>,t+1}$, $E_{<i,j>,t} \subseteq E_{<i,j>,t+1}$ and $P_{<i,j>,k,t} \subseteq P_{<i,j>,k,t+1}$ for each $k \in \mathbb{N}$. For each $e \in \mathbb{N}$, let $\mathcal{E}_e = \bigcup_t \mathcal{E}_{e,t}$. The sequence $\mathcal{E}_0, \mathcal{E}_1, \cdots$ lists all computable weighted equivalence structures. ∎

To ensure that the weighted equivalence structure $\mathcal{F}$ is computably categorical, we satisfy the following requirements for all $e \in \mathbb{N}$:

$$R_e : \mathcal{E}_e \ncong \mathcal{F} \text{ or } \mathcal{E}_e \text{ is computably isomorphic to } \mathcal{F}.$$

The construction will be carried out in stages. At stage t, we will construct a finite weighted equivalence structure $\mathcal{F}_t$ such that $\mathcal{F}_0 \subset \mathcal{F}_1 \subset \mathcal{F}_2 \subset \ldots$ and set $\mathcal{F} = \bigcup_t \mathcal{F}_t$.

6.5.2 Construction of $\mathcal{F}$

The construction uses the tree argument (See [108] for a detailed introduction). Intuitively, we construct $\mathcal{F}$ by putting all strategies on the binary tree $T = 2^{<\omega}$. We satisfy all requirements by traversing the tree T along paths of the tree. For each node we visit, we carry out the construction for satisfying one requirement. We use lower case Greek letters $\alpha, \beta, \gamma, \ldots$ to denote nodes on T, i.e., finite strings over $\{0,1\}$. The tree order on T is the prefix order $\leq_{\text{pref}}$. A node α is *to the left* of another node β, denoted by $\alpha <_{\text{L}} \beta$ if there is $\gamma \in T$ such that $\gamma 0 \leq_{\text{pref}} \alpha$ and $\gamma 1 \leq_{\text{pref}} \beta$. Let $|\alpha|$ denote the length of α. A *path* is a (possibly infinite) sequence of strings $\alpha_0, \alpha_1, \alpha_2, \ldots \in \{0,1\}^\star$ such that $\alpha_0 = \varepsilon$ and for all $i \in \mathbb{N}$, $\alpha_{i+1} \in \{\alpha_i 0, \alpha_i 1\}$.

Hence a finite path can also be identified by its last node. For a path δ in T and $n \in \mathbb{N}$, we let $\delta \upharpoonright n$ denote the length n initial segment of δ, i.e., the level n node on the path δ. We use $\delta(n)$ to denote the symbol $i \in \{0,1\}$ such that $\delta \upharpoonright n+1 = (\delta \upharpoonright n)i$.

When we visit a node α, we try to satisfy the requirement $R_{|\alpha|}$ by defining a set of *waiting* components in $\mathcal{F}$. We search in the structure $\mathcal{E}_{|\alpha|}$ for components that are isomorphic to these waiting components. If such components are found, we match them isomorphically with the waiting components in $\mathcal{F}$ and declare that these waiting components are now *covered by* α. We then set some *other* components as the new waiting components. In this case, we say that the requirement $R_{|\alpha|}$ *recovers*. If no such component is found, $R_{|\alpha|}$ keeps *waiting*. The path we traverse in the tree T depends on the outcomes we obtain for each $\alpha \in T$:

- If $R_{|\alpha|}$ recovers, we next visit the node $\alpha 0$ and act to satisfy the next requirement $R_{|\alpha 0|}$. All waiting components for $\alpha 0$ are taken from the covered components for α.
- If $R_{|\alpha|}$ does not recover, we next visit the node $\alpha 1$ and act to satisfy $R_{|\alpha 1|}$ by picking waiting components for $\alpha 1$ from the other components for α.

Formally, for each node $\alpha \in T$ and $t \in \mathbb{N}$, we define the tuple

$$(\mathcal{C}_{\alpha,t}, \mathcal{W}_{\alpha,t}, \mathcal{O}_{\alpha,t}, \mathcal{C}^{\mathrm{M}}_{\alpha,t}, \mathcal{W}^{\mathrm{M}}_{\alpha,t}, \mathcal{O}^{\mathrm{M}}_{\alpha,t}, M_{\alpha,t}, A_{\alpha,t})$$

such that

- $\mathcal{C}_{\alpha,t}, \mathcal{W}_{\alpha,t}, \mathcal{O}_{\alpha,t}$ are pairwise disjoint sets of components in $\mathcal{F}_t$
- $M_{\alpha,t}, A_{\alpha,t}$ are two (distinct) components that do not belong to $\mathcal{C}_{\alpha,t} \cup \mathcal{W}_{\alpha,t} \cup \mathcal{O}_{\alpha,t}$.
- For all $\mathcal{K} \in \{\mathcal{C}, \mathcal{W}, \mathcal{O}\}$, $\mathcal{K}^{\mathrm{M}}_{\alpha,t} \subseteq \mathcal{K}_{\alpha,t}$ and $|\mathcal{W}^{\mathrm{M}}_{\alpha,t}| = 1$.

The components in $\mathcal{C}_{\alpha,t}, \mathcal{W}_{\alpha,t}, \mathcal{O}_{\alpha,t}$ are respectively called the *covered*, *waiting* and *other* components for α at stage t. We say that the component $M_{\alpha,t}$ is *marked by* α at stage t and $A_{\alpha,t}$ is *reserved*. Since $\mathcal{W}^{\mathrm{M}}_{\alpha,t}$ is a singleton, we sometimes abuse the notation by treating it as a component. The set $\mathcal{C}^{\mathrm{M}}_{\alpha,t}$ contains components that have been marked by α at a prior stage. The set $\mathcal{O}^{\mathrm{M}}_{\alpha,t}$ contains the components that may be marked by α in future stages.

We maintain these sets of components in such a way that the following properties hold; See Figure 6.2 for an illustration:

$$\begin{aligned}
\mathcal{C}_{\alpha 0,t} \cup \mathcal{W}_{\alpha 0,t} \cup \mathcal{O}_{\alpha 0,t} \cup \{M_{\alpha 0,t}, A_{\alpha 0,t}\} &= \mathcal{C}_{\alpha,t} \\
\mathcal{C}_{\alpha 1,t} \cup \mathcal{W}_{\alpha 1,t} \cup \mathcal{O}_{\alpha 1,t} \cup \{M_{\alpha 1,t}, A_{\alpha 1,t}\} &= \mathcal{O}_{\alpha,t} \\
\mathcal{C}^{\mathrm{M}}_{\alpha 0,t} \cup \mathcal{W}^{\mathrm{M}}_{\alpha 0,t} \cup \mathcal{O}^{\mathrm{M}}_{\alpha 0,t} \cup \{M_{\alpha 0,t}, A_{\alpha 0,t}\} &\subseteq \mathcal{C}^{\mathrm{M}}_{\alpha,t} \\
\mathcal{C}^{\mathrm{M}}_{\alpha 1,t} \cup \mathcal{W}^{\mathrm{M}}_{\alpha 1,t} \cup \mathcal{O}^{\mathrm{M}}_{\alpha 1,t} \cup \{M_{\alpha 1,t}, A_{\alpha 1,t}\} &\subseteq \mathcal{O}^{\mathrm{M}}_{\alpha,t}
\end{aligned}$$

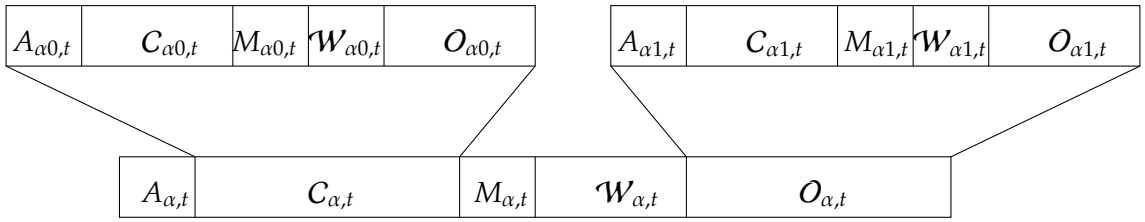


Figure 6.2: Components in the sets $C_{\alpha,t}, \mathcal{W}_{\alpha,t}, O_{\alpha,t}, M_{\alpha,t}, A_{\alpha,t}$

To maintain these properties, at any stage t, we make sure the following for all components C and all $\alpha \in T$:

- If $C \in C_{\alpha,t}$ then $C \in O_{\alpha 0,t}$. In particular, $C \in O^{\mathrm{M}}_{\alpha 0,t}$ if $C \in C^{\mathrm{M}}_{\alpha,t}$.
- If $C \in O_{\alpha,t}$ then $C \in O_{\alpha 1,t}$. In particular, $C \in O^{\mathrm{M}}_{\alpha 1,t}$ if $C \in O^{\mathrm{M}}_{\alpha,t}$.
- C can only be set as $\mathcal{W}^{\mathrm{M}}_{\alpha,t}$ or $A_{\alpha,t}$ after it is put in $O^{\mathrm{M}}_{\alpha,t}$.
- C can only be marked by α after it is set as $\mathcal{W}^{\mathrm{M}}_{\alpha,t}$.
- C can only be a member of $C^{\mathrm{M}}_{\alpha,t}$ only after it has been marked by α.

We denote by $\mathcal{F}_{\alpha,t}$ the substructure obtained by taking the disjoint union of all components in $C_{\alpha,t} \cup \mathcal{W}_{\alpha,t} \cup O_{\alpha,t} \cup \{M_{\alpha,t}, A_{\alpha,t}\}$. During the construction, we define for each node $\alpha \in T$ a partial isomorphism $f_{\alpha,t}$ from $\mathcal{E}_{|\alpha|,t}$ to $\mathcal{F}_{\alpha,t}$. We say a component C is *mapped by α at stage t* if $f^{-1}_{\alpha,t}(x)$ is defined for some $x \in C$. An element x is *unique* if no other element has the same weight as x. A number n is *unused* if no element in the currently constructed $\mathcal{F}$ has weight n. During the construction we maintain the following additional invariants for all $\alpha \in T$ and $t \in \mathbb{N}$:

(S1) $\forall \beta >_{\mathrm{pref}} \alpha : A_{\alpha,t} \prec A_{\beta,t}$.

(S2) Every component in $O_{\varepsilon,t} \setminus O^{\mathrm{M}}_{\varepsilon,t}$ has a unique element in $\mathcal{F}_t$.

(S3) At stage t, all components in $C_{\alpha,t}$ are mapped by α while no component in $\mathcal{W}_{\alpha,t} \cup O_{\alpha,t}$ is mapped by α.

At stage t of the construction, for any $\alpha \in T$ and component $C \in C_{\alpha,t} \cup \{M_{\alpha,t}\} \cup \mathcal{W}_{\alpha,t}$, we will pick a number m^C_α such that $\mathbf{m}^{\mathbf{C}}_\alpha \in C$ and m^C_α is distinct for all C and α, i.e., for all components C, D in $\mathcal{F}_t$ and $\alpha, \beta \in T$, $m^C_\alpha = m^D_\beta$ implies $C = D$ and $\alpha = \beta$. We also maintain the following invariants for each t and $\alpha \in T$:

(S4) For $C \in (\mathcal{C}_{\alpha,t} \setminus \mathcal{C}^{\mathrm{M}}_{\alpha,t}) \cup \{M_{\alpha,t}\} \cup \mathcal{W}_{\alpha,t}$, $\mathbf{m}^{\mathbf{C}}_{\alpha}$ is unique in $\mathcal{F}_t$; For $C \in \mathcal{C}^{\mathrm{M}}_{\alpha,t}$, $\mathbf{m}^{\mathbf{C}}_{\alpha}$ is only contained in components in $\mathcal{C}^{\mathrm{M}}_{\alpha,t} \cup \{M_{\alpha,t}\}$. Furthermore, $f^{-1}_{\alpha,t}$ is defined on the element with weight m^C_α in C for any $C \in \mathcal{C}_{\alpha,t}$.

For $\alpha \in T$ and $t \in \mathbb{N}$, define

$$\mathsf{Check}_{\alpha,t} = \{m^C_\alpha \mid C \in \mathcal{C}^{\mathrm{M}}_{\alpha,t} \cup \mathcal{W}^{\mathrm{M}}_{\alpha,t} \cup \{M_{\alpha,t}\}\}.$$

With some nodes α in the tree, we associate a number Forbid_α as follows: At stage t, for each α we visit and each $n \in \mathsf{Check}_{\alpha,t}$, if the number of components in $\mathcal{E}_{|\alpha|,t}$ that contains $\mathbf{n}$ is more than the number of components in $\mathcal{F}_t$ that contains $\mathbf{n}$, we set $\mathsf{Forbid}_\alpha = n$. A number n is *forbidden* whenever $n = \mathsf{Forbid}_\alpha$ for some $\alpha \in T$. Intuitively speaking, if n is forbidden, no further element with weight n is allowed to be created in $\mathcal{F}$.

At stage t of the construction, we may *initialize* a node α by applying the following operations:

(1) Undefine the number m^C_α for all component $C \in \mathcal{F}_{\alpha,t}$.

(2) Set $\mathcal{F}_{\alpha,t}$ empty (In particular, set all of $\mathcal{C}_{\alpha,t}, \mathcal{W}_{\alpha,t}, \mathcal{O}_{\alpha,t}, A_{\alpha,t}$ and $M_{\alpha,t}$ empty).

(3) If Forbid_α has been defined and is equal to some $n \in \mathbb{N}$, then undefine it. Hence the number n is no longer forbidden.

Construction. We now describe the stagewise construction. At stage 0, we initialize all nodes $\alpha \in T$.

At stage $t+1$, we first set $K_{\alpha,t+1} = K_{\alpha,t}$ for all $K \in \{\mathcal{C}, \mathcal{W}, \mathcal{O}, A, M, f\}$, $K^{\mathrm{M}}_{\alpha,t+1} = K^{\mathrm{M}}_{\alpha,t}$ for all $K \in \{\mathcal{C}, \mathcal{W}, \mathcal{O}\}$, then run a sequence of *steps* to construct the structure $\mathcal{F}_{t+1}$. The path δ_{t+1} is defined inductively on the steps. The number of steps at stage $t+1$ is at most $t+1$ and hence $|\delta_{t+1}| \leq t+1$. Let $\delta_{t+1} \upharpoonright 0 = \varepsilon$. Suppose we are at step $s < t+1$ and $\alpha = \delta_{t+1} \upharpoonright s$ is defined. The construction acts on the node α and s may either be an *α-recovery step* or an *α-waiting step*, which is classified by the following algorithm:

1. If Forbid_α is defined, s is an α-waiting step.

2. Suppose Forbid_α is not defined. For each $n \in \mathsf{Check}_\alpha$, if the number of components containing $\mathbf{n}$ in $\mathcal{E}_{|\alpha|,t+1}$ is more than the number of components containing $\mathbf{n}$ in $\mathcal{F}_t$, then we set $\mathsf{Forbid}_\alpha = n$ and s is an α-waiting step.

3. If $\mathcal{O}^{\mathrm{M}}_{\alpha,t+1} = \emptyset$, then stop this step and do not carry out any more steps in this stage.

4. If $A_{\alpha,t+1}$ is undefined, set $A_{\alpha,t+1}$ as the first component[3] C in $\mathcal{O}^{\mathrm{M}}_{\alpha,t+1}$, and delete C from $\mathcal{O}_{\alpha,t+1}$. If $s > 0$, add to $A_{\alpha,t+1}$ a new element $\mathbf{n}$ if $\mathbf{n} \in A_{\delta_{t+1}\restriction(s-1),t+1}$ and $\mathbf{n} \notin A_{\alpha,t+1}$. Note that this operation is allowed only when n is not forbidden for any $\mathbf{n} \in A_{\delta_{t+1}\restriction(s-1),t+1}$. This fact is proved in Lemma 6.5.7(a).

5. If $M_{\alpha,t+1}$ is undefined and $\mathcal{W}^{\mathrm{M}}_{\alpha,t} \neq \emptyset$, then set $M_{\alpha,t+1}$ as $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ and set $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1} = \emptyset$.

6. If $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1} = \emptyset$ and $\mathcal{O}^{\mathrm{M}}_{\alpha,t+1} \neq \emptyset$ (note that $\mathcal{O}^{\mathrm{M}}_{\alpha,t+1}$ may have been updated in 4.), then set $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ as the first component C in $\mathcal{O}^{\mathrm{M}}_{\alpha,t+1}$ (hence delete it from $\mathcal{O}^{\mathrm{M}}_{\alpha,t+1}$). To C add a new element with unused weight m and set $m^C_\alpha = m$.

7. If any one of $M_{\alpha,t+1}$ and $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ is not defined, then s is an α-waiting step.

8. Suppose $M_{\alpha,t+1}$ and $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ are both defined. If $f_{\alpha,t}$ can be extended to a partial isomorphism from $\mathcal{E}_{|\alpha|,t+1}$ to $\mathcal{F}_t$ such that it maps components in $\mathcal{E}_{|\alpha|,t+1}$ isomorphically to $\{M_{\alpha,t+1}\} \cup \mathcal{W}_{\alpha,t+1}$, then s is an α-recovery step. Otherwise it is an α-waiting step.

Define the next node on the path δ_{t+1} as

$$\delta_{t+1}(s) = \begin{cases} 0 & \text{if } s \text{ is an } \alpha\text{-recovery step} \\ 1 & \text{otherwise.} \end{cases}$$

If $\delta_{t+1}(s) = 1$, then we proceed to the next step. Otherwise, we act for α as follows. Firstly, we extend $f_{\alpha,t}$ such that it maps components in $\mathcal{E}_{|\alpha|,t}$ isomorphically to components in $\{M_{\alpha,t}\} \cup \mathcal{W}_{\alpha,t}$. Then we perform the following operations:

1. Move all components in $\{M_{\alpha,t+1}\} \cup \mathcal{W}_{\alpha,t+1} \setminus \mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ to $\mathcal{C}_{\alpha,t+1}$. In particular, move $M_{\alpha,t+1}$ to $\mathcal{C}^{\mathrm{M}}_{\alpha,t+1}$.

2. Add all components in $\{M_{\alpha,t+1}\} \cup \mathcal{W}_{\alpha,t+1} \setminus \mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ to $\mathcal{O}_{\alpha 01^m,t+1}$ for all $m \in \mathbb{N}$. In particular, add $M_{\alpha,t+1}$ to $\mathcal{O}^{\mathrm{M}}_{\alpha 01^m,t+1}$.

3. To $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ add an element with weight n if $\mathbf{n} \in M_{\alpha,t+1}$ and $\mathbf{n} \notin \mathcal{W}^{\mathrm{M}}_{\alpha,t}$. Note that this operation is allowed only if for all $\mathbf{n} \in M_{\alpha,t+1}$, n is not forbidden. This fact is proved in Lemma 6.5.7(b).

4. Set the new $M_{\alpha,t+1}$ as $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ and set $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1} = \emptyset$.

5. To each component $C \in \mathcal{O}^{\mathrm{M}}_{\alpha,t+1}$ add an element with unused weight so that they are pairwise non-embeddable. Set m^C_α as the weight of the new element.

[3] We assume there is a well-order on all components in $\mathcal{F}_t$, e.g., a component C is less than another component D if the least element in C is smaller than the least element in D (recall that elements in $\mathcal{F}$ are natural numbers). By the "first" component, we mean the least with respect to this linear order.

6. For each component $C \in \mathcal{O}_{\alpha,t+1} \setminus \mathcal{O}^{\mathrm{M}}_{\alpha,t+1}$, if $C \in \mathcal{O}_{\beta,t+1}$ for all $\beta \leq_{\mathrm{pref}} \alpha$, then let m^C_α be the unique element contained in C (such element exists by (S2)); otherwise, let s' be the largest number such that $C \notin \mathcal{O}_{\delta_{t+1}\upharpoonright s',t+1}$. It must be that $C \in \mathcal{C}_{\delta_{t+1}\upharpoonright s',t+1}$, and thus $m^C_{\delta_{t+1}\upharpoonright s'}$ is defined. Let $m^C_\alpha = m^C_{\delta_{t+1}\upharpoonright s'}$.

7. Move all components in $\mathcal{O}_{\alpha,t}$ to $\mathcal{W}_{\alpha,t+1}$. In particular, set $\mathcal{W}^{\mathrm{M}}_{\alpha,t+1}$ as the first component in $\mathcal{O}^{\mathrm{M}}_{\alpha,t}$.

8. Initialize all nodes $\beta \in \alpha 1\{0,1\}^\star$.

This completes the construction for step s. The following lemma shows that no element created in $\mathcal{F}_{t+1}$ has a forbidden weight.

Lemma 6.5.7 *The following statements hold at step s of stage $t+1$:*

(a) Suppose $s > 0$ and $A_{\alpha,t+1}$ is not defined at step $s-1$. Then for all $\mathbf{n} \in A_{\delta_{t+1}\upharpoonright(s-1),t+1}$, n is not forbidden.

(b) Suppose s is an α-recovery step. For all $\mathbf{n} \in M_{\alpha,t}$, n is not forbidden.

Proof. (a) Let $\beta = \delta_{t+1} \upharpoonright s-1$. Take $\mathbf{n} \in A_{\beta,t+1}$ and suppose $n = \mathsf{Forbid}_\gamma$ for some $\gamma \in T$. This implies that $n = m^C_\gamma$ for some component

$$C \in \mathcal{C}^{\mathrm{M}}_{\gamma,t+1} \cup \mathcal{W}^{\mathrm{M}}_{\gamma,t+1} \cup \{M_{\gamma,t+1}\}.$$

We have the following cases:

(1) Suppose $\gamma \geq_{\mathrm{pref}} \alpha$. By construction, for all $\tau \in T$, the set $\mathcal{C}_{\tau,t+1} \cup \{M_{\tau,t+1}\} \cup \mathcal{W}_{\tau,t+1}$ is nonempty only if $A_{\tau,t+1}$ is defined. Since $A_{\alpha,t+1}$ is not defined, $\mathcal{C}_{\gamma,t+1} \cup \{M_{\gamma,t+1}\} \cup \mathcal{W}_{\gamma,t+1}$ is empty and Forbid_γ is undefined. Contradiction.

(2) Suppose $\gamma >_{\mathrm{L}} \beta$. The construction would have initialized γ at stage $t+1$ and γ has not been visited again. Therefore we also have Forbid_γ undefined.

(3) Suppose $\gamma \leq_{\mathrm{L}} \beta$. By (S4), $m^C_\gamma \notin D$ for any $D \notin \mathcal{C}_{\gamma,t+1} \cup \{M_{\gamma,t+1}\} \cup \mathcal{W}_{\gamma,t+1}$. Since $A_{\beta,t+1} \notin \mathcal{C}_{\gamma,t+1} \cup \{M_{\gamma,t+1}\} \cup \mathcal{W}_{\gamma,t+1}$, we have $\mathbf{n} \notin A_{\beta,t+1}$, which is in contradiction with the assumption.

(4) Suppose $\gamma <_{\mathrm{pref}} \beta$. Then $\gamma = \delta_{t+1} \upharpoonright s'$ for some $s' < s-1$. This means that s' is a $\delta_{t+1} \upharpoonright s'$-waiting step and $A_{\beta,t+1} \in \mathcal{O}_{\delta_{t+1}\upharpoonright s',t+1}$. Thus $\mathbf{n} \notin A_{\beta,t+1}$. Contradiction.

Since we arrive at a contradiction in all cases above, (a) is proved.

(b) Take $\mathbf{n} \in M_{\alpha,t}$ and suppose n is forbidden before step s. Since all nodes $\gamma >_{\mathrm{L}} \alpha$ has been initialized at stage $t+1$, $n = m^{C}_{\beta}$ for some $\beta \not>_{\mathrm{L}} \alpha$ and some component

$$C \in C^{\mathrm{M}}_{\beta,t+1} \cup \mathcal{W}^{\mathrm{M}}_{\beta,t+1} \cup \{M_{\beta,t+1}\}.$$

We have the following cases:

(1) Suppose $\beta >_{\mathrm{pref}} \alpha$. By (S4), $\mathbf{m}^{\mathbf{C}}_{\beta} \notin D$ for any $D \notin C_{\beta,t+1} \cup \{M_{\beta,t+1}\} \cup \mathcal{W}_{\beta,t+1}$. But $F_{\beta,t+1}$ is formed by either components in $C_{\alpha,t+1}$ (when $\beta \geq_{\mathrm{pref}} \alpha 0$) or $O_{\alpha,t+1}$ (when $\beta \geq_{\mathrm{pref}} \alpha 1$). Hence $M_{\alpha,t+1} \notin C_{\beta,t} \cup \{M_{\beta,t}\} \cup \mathcal{W}_{\beta,t}$, which implies $\mathbf{m}^{\mathbf{C}}_{\beta} \notin M_{\alpha,t}$.

(2) Suppose $\beta = \alpha$. In this case, $\mathsf{Forbid}_{\alpha} \neq \emptyset$ and s would have be an α-waiting step, which is in contradiction with the assumption.

(3) Suppose $\beta <_{\mathrm{L}} \alpha$. Then there is $\gamma <_{\mathrm{pref}} \beta$ such that $M_{\alpha,t+1} \in O_{\gamma,t+1}$. This means that $M_{\alpha,t+1} \notin C_{\beta,t+1} \cup \{M_{\beta,t+1}\} \cup \mathcal{W}_{\beta,t+1}$. Hence $m^{C}_{\beta} \notin M_{\alpha,t+1}$.

(4) Suppose $\beta <_{\mathrm{pref}} \alpha$. There is a step $s' < s$ at stage $t+1$ such that $\mathsf{Forbid}_{\delta_{t+1} \restriction s'} = n$. Therefore s' is a $\delta_{t+1} \restriction s'$-waiting step. This means that $M_{\alpha,t+1} \in O_{\delta_{t+1} \restriction s',t+1}$ and thus $\mathbf{n} \notin M_{\alpha,t+1}$.

Since we arrive at a contradiction in all cases above, (b) is proved. ∎

After finishing all steps, we create new components $N_1 = \{\mathbf{n_1}\}, N_2 = \{\mathbf{n_2}\}$, where n_1, n_2 are distinct and unused weights. Add N_1 and N_2 to $O_{1^m,t+1}$ for all $m \in \mathbb{N}$. In particular, add N_2 to $O^{\mathrm{M}}_{1^m,t+1}$. This completes the construction of $\mathcal{F}_{t+1}$.

6.5.3 Verification

We now prove that the invariants are preserved by the construction. Suppose all invariants hold at stage t.

Lemma 6.5.8 *The invariants (S1) – (S4) hold at stage $t+1$.*

Proof. Fix $\alpha \in T$. We prove that all invariants hold for α at stage $t+1$.

(S1) : Whenever a component is set as $A_{\alpha,t+1}$ for some α with $|\alpha| > 0$, the construction will add into $A_{\alpha,t+1}$ any element $\mathbf{n}$ if $\mathbf{n} \in A_{\delta \restriction |\alpha|-1,t+1}$ and $\mathbf{n} \notin A_{\alpha,t+1}$. Also, by the construction, once a component is set as $A_{\alpha,t+1}$, it will no longer be extended in the future. Hence for all $\beta \in T$, $\alpha <_{\mathrm{pref}} \beta$ implies $A_{\alpha,t} \prec A_{\beta,t}$.

(S2) : By the construction once a component is an element of $O_{\alpha,s} \setminus O^{\mathrm{M}}_{\alpha,s}$ for some $\alpha \in T$ and $s \in \mathbb{N}$, it will never be extended in the future. Note also that the set of components $O_{\varepsilon,t+1} \setminus O^{\mathrm{M}}_{\varepsilon,t+1}$ is a subset of $(O_{\varepsilon,t} \setminus O^{\mathrm{M}}_{\varepsilon,t}) \cup N_1$. Hence (S2) holds by assumption and the fact that N_1 contains an unique element.

(S3) : This invariant holds at stage $t+1$ because a component C is moved to $C_{\alpha,t+1}$ only when $f_{\alpha,t+1}$ maps some element in $\mathcal{F}_{t+1}$ to C.

(S4) : Take a component $C \in (C_{\alpha,t+1} \setminus C^{\mathrm{M}}_{\alpha,t+1}) \cup \{M_{\alpha,t+1}\} \cup \mathcal{W}_{\alpha,t+1}$. We prove, by induction on the number of steps performed, that the element $\mathbf{m}^{\mathbf{C}}_{\alpha}$ contained in C is unique in $\mathcal{F}_{t+1}$. Suppose $\mathbf{m}^{\mathbf{C}}_{\alpha}$ is unique (if it is defined) at step $s-1$. Let $\beta = \delta_{t+1} \upharpoonright s$. At step s, there are two cases where an element with a used weight could be created:

(i) If $A_{\beta,t+1}$ is undefined before the step and $O^{\mathrm{M}}_{\beta,t+1}$ is not empty, a component in $O^{\mathrm{M}}_{\beta,t+1}$ is selected as the new $A_{\beta,t+1}$ and the construction adds to $A_{\beta,t+1}$ all elements $\mathbf{n}$ where $\mathbf{n} \in A_{\delta_{t+1}\upharpoonright s-1,t+1}$. We prove below that $A_{\delta_{t+1}\upharpoonright s-1,t+1}$ does not contain $\mathbf{m}^{\mathbf{C}}_{\alpha}$. If $\alpha \not<_{\mathrm{pref}} \beta$, then it is clear that $A_{\delta_{t+1}\upharpoonright s-1,t+1} \notin C_{\alpha,t+1} \cup \{M_{\alpha,t+1}\} \cup \mathcal{W}_{\alpha,t+1}$. If $\alpha <_{\mathrm{pref}} \delta_{t+1} \upharpoonright s-1$, $A_{\delta_{t+1}\upharpoonright s-1,t+1}$ either belongs to $C^{\mathrm{M}}_{\alpha,t+1}$ or $O^{\mathrm{M}}_{\alpha,t+1}$ and hence does not contain $\mathbf{m}^{\mathbf{C}}_{\alpha}$ by the inductive hypothesis. If $\alpha = \delta_{t+1} \upharpoonright s-1$, then also by the inductive hypothesis, $A_{\alpha,t+1}$ clearly does not contain $\mathbf{m}^{\mathbf{C}}_{\alpha}$.

(ii) If s is a β-recovery stage, then the construction sets the component $\mathcal{W}^{\mathrm{M}}_{\beta,t+1}$ as the new β-marked component and adds to it all elements $\mathbf{n}$ that are contained in the previous β-marked component. We use M to denote the previous β-marked component. Using a similar argument as in (i), one can prove that M does not contain $\mathbf{m}^{\mathbf{C}}_{\alpha}$.

Now take $C \in C^{\mathrm{M}}_{\alpha,t+1}$. By construction there is a stage $t' \leq t+1$ where C is marked by α at stage t'. Then a component contains $\mathbf{m}^{\mathbf{C}}_{\alpha,\mathbf{t+1}}$ if and only if it is marked by α after stage t'. Hence all components containing $\mathbf{m}^{\mathbf{C}}_{\alpha,\mathbf{t+1}}$ belong to $C^{\mathrm{M}}_{\alpha,t+1} \cup \{M_{\alpha,t+1}\}$.

Note also that by construction, as a component C is moved to $C_{\alpha,t+1}$, $f^{-1}_{\alpha,t}$ is defined on the element with weight m^{C}_{α} in C. ∎

The next lemma shows that the construction visits every level of the tree infinitely often.

Lemma 6.5.9 *For every $e \in \mathbb{N}$, there is some $\alpha \in T$ such that $|\alpha| = e$ and α is visited by infinitely many δ_s.*

Proof. We prove the lemma by induction on the stages. The base case is when $e = 0$ and $\alpha = \varepsilon$. The inductive hypothesis assumes that for $e \geq 0$ there is $\alpha \in T$ where $|\alpha| = e$ and for infinitely many stages t, we have (1) $\alpha \in \delta_t$ and (2) $O_{\alpha,t} \neq \emptyset$. We have two cases:

Case 1: If $R_{|\alpha|}$ recovers at α infinitely often, then $\alpha 0$ is visited infinitely often. At any stage s where $\alpha 0 \in \delta_s$, the construction will move the marked and waiting components for α to $O_{\alpha 0,s}$. Hence the inductive hypothesis holds for $e+1$.

Case 2: If after some stage, $R_{|\alpha|}$ would never recover, then $\alpha 1$ is visited infinitely often. Let γ be the shortest word such that $\alpha = \gamma 1^m$ for some $m \in \mathbb{N}$. Since γ is visited infinitely often,

for infinitely many stages s, some new components are added to $O_{\gamma,s}$. Since $\alpha = \gamma 1^m \in \delta$, there is a stage s after which all nodes in $\{\gamma 1^k \mid k \leq m\}$ are never initialized. Therefore at all stages $s' > s$ there are some components put in $O_{\beta,s'}$. Hence the inductive hypothesis holds for $e+1$. ∎

We define the *true path* δ such that for all $e \in \mathbb{N}$, $\delta \upharpoonright e = \liminf_s \delta_s \upharpoonright e$. In other words, for any $e \in \mathbb{N}$, $\delta \upharpoonright e$ is the leftmost γ such that $|\gamma| = e$ and $\exists^\infty s : \gamma \in \delta_s$. By Lemma 6.5.9, the true path δ exists. For all $e \in \mathbb{N}$, let s_e be the least stage after which the construction would never initialize the node $\delta \upharpoonright e$. Let $A_e = A_{\delta\upharpoonright e, s_e}$.

Lemma 6.5.10 *The structure $\mathcal{F}$ contains an infinite chain of properly embedded components. The set of elements whose components properly embed into infinitely many components is computable in $0''$*

Proof. By (S1), for all e, $A_e \prec A_{e+1}$. Hence the sequence $A_0, A_1, A_2, \ldots$ forms an infinite chain of properly embedded components in $\mathcal{F}$. Note also that the set of elements whose components properly embed into infinitely many components are exactly the elements in $\bigcup_e A_e$. Since the true path δ satisfies that $\forall e : \delta \upharpoonright e = \liminf_t \delta_t \upharpoonright e$, δ is computable in $0''$. For each $e \in \mathbb{N}$, using a $0''$-oracle, we are able to compute the least stage s_e after which $\delta \upharpoonright e$ is never initialized. Running the construction till stage s_e reveals the component $A_{\delta\upharpoonright e, s_e} = A_e$. ∎

The next lemma shows that, intuitively speaking, the true path δ indicates which $\mathcal{E}_e$ is isomorphic to $\mathcal{F}$.

Lemma 6.5.11 *For all $e \in \mathbb{N}$, $\mathcal{E}_e \cong \mathcal{F}$ implies $\delta(e) = 0$.*

Proof. Suppose that $\mathcal{E}_e \cong \mathcal{F}$ and there is a stage t after which R_e never recovers, i.e., $\delta(e) = 1$. Let $\alpha = \delta \upharpoonright e$. Without loss of generality, assume α is never initialized after stage t. Thus $\mathcal{W}_{\alpha,t} = \mathcal{W}_{\alpha,s}$ for all $s > t$. Take a component $H \in \mathcal{W}_{\alpha,t} \cup \{M_{\alpha,t}\}$. By (S4), for all $s > t$, H has a unique element $\mathbf{m}^{\mathbf{H}}_\alpha$ in $\mathcal{F}_s$. Therefore H is the only component in $\mathcal{F}$ that contains $\mathbf{m}^{\mathbf{H}}_\alpha$. Since $\mathcal{E}_e \cong \mathcal{F}$, in $\mathcal{E}_e$ there is a unique element C_H that is isomorphic to H. We prove next that before stage s, the function $f_{\alpha,t}$ is not defined on C_H and hence the construction will eventually find C_H in $\mathcal{E}_e$. This is sufficient to prove the lemma as R_e would recover at α when C_H is found for all $H \in \{M_{\alpha,t}\} \cup \mathcal{W}_{\alpha,t}$, which is in contradiction with the assumption.

Suppose $H \in \mathcal{W}_{\alpha,t}$. By (S4), every component $C \in \mathcal{C}_{\alpha,t}$ has some element $\mathbf{n}$ that is not contained in H and $f^{-1}_{\alpha,t}$ is defined on $\mathbf{n}$. This means that the component $f^{-1}_{\alpha,t}(C)$ in $\mathcal{E}_e$ is not isomorphic to H. Therefore $f_{\alpha,t}$ is not defined on C_H.

Suppose $H = M_{\alpha,t}$. Then for all $t' < t$, $M_{\alpha,t'} \prec M_{\alpha,t}$. Suppose for the sake of contradiction that there is a stage $t' < t$ such that $f_{\alpha,t'}$ maps the component C_H to $M_{\alpha,t'}$. Let s be the last stage where R_e recovers at α before t. Note that $\mathcal{W}^{\mathrm{M}}_{\alpha,s}$ and $M_{\alpha,t}$ are the same component.

By the definition of an α-recovery stage, $f^{-1}_{\alpha,s}(\mathcal{W}^{\mathrm{M}}_{\alpha,s})$ contains $\mathbf{m}_\alpha^{\mathcal{W}^{\mathrm{M}}_{\alpha,\mathbf{s}}} = \mathbf{m}^{\mathbf{H}}_\alpha$. Let $\ell \geq s$ be the first stage when C_H completely reveals itself. At stage ℓ, both components C_H and $f^{-1}(H)$ in $\mathcal{E}_{e,\ell}$ contain an element with weight m^H_α, whereas only one component H in F_ℓ contains an element with weight m^H_α. Furthermore, it is clear that $m^H_\alpha \in \mathsf{Check}_{\alpha,\ell}$. Therefore at stage ℓ, the construction would set $\mathsf{Forbid}_\alpha = m^H_\alpha$ and guarantee that $\mathcal{E}_e \not\cong \mathcal{F}$. This is in contradiction with the assumption. Therefore, $f_{\alpha,t}$ is not defined on C_H. ∎

The next lemma shows that all requirements R_e, $e \in \mathbb{N}$, are satisfied. Let $\mathcal{F}(e)$ be the structure obtained by the union

$$\bigcup_{s \geq s_e} C_{\delta\restriction e,s} \cup \{M_{\delta\restriction e,s}\} \cup \mathcal{W}_{\delta\restriction e,s} \cup \mathcal{O}_{\delta\restriction e,s}.$$

Since $\delta \restriction e$ is never initialized after s_e and is visited infinitely often, the structure $\mathcal{F}(e)$ contains all but finitely many components in $\mathcal{F}$

Lemma 6.5.12 *If $\mathcal{E}_e \cong \mathcal{F}$, then $\mathcal{E}_e$ and $\mathcal{F}$ are computably isomorphic. Hence the requirement R_e for all $e \in \mathbb{N}$ are satisfied.*

Proof. Suppose $\mathcal{E}_e \cong \mathcal{F}$. By Lemma 6.5.11, $\delta(e) = 0$ and R_e recovers infinitely often at $\alpha = \delta \restriction e$. Hence by construction, all components in $\mathcal{F}(e)$ are eventually covered by f_α, i.e.,

$$\mathcal{F}(e) = \bigcup_{s \geq s_e} C_{\delta\restriction e,s}. \tag{6.1}$$

By (S3) this means that f_α is eventually defined on all components in $\mathcal{F}(e)$. We define a mapping f as follows:

- First non-uniformly map the components not in $\mathcal{F}(e)$ with their corresponding isomorphic copies in $\mathcal{E}_e$.
- Then extend f by the function $f_\alpha = \bigcup_s f_{\alpha,s}$.

We now prove that f is indeed an isomorphism from $\mathcal{E}_e$ to $\mathcal{F}$. Let H be a component of $\mathcal{F}(e)$. By (6.1), $H \in C_{\alpha,t}$ for some $t \geq s_e \in \mathbb{N}$ There is a stage $s > t$ where $H \in \mathcal{W}_{\alpha,s}$. We have the following two cases:

- Case 1. The component H is never marked by α. At stage s, by (S4), H contains an element $\mathbf{m}^{\mathbf{H}}_\alpha$ that is unique in $\mathcal{F}_s$. Since H is never marked by α, it does not belong to $C^{\mathrm{M}}_{\alpha,s'}$ for all $s' \geq s$. By (S4), for all $s' \geq s$, H is the only component containing $\mathbf{m}^{\mathbf{H}}_\alpha$ in $\mathcal{F}_s$. Hence H is the only component containing $\mathbf{m}^{\mathbf{H}}_\alpha$ in $\mathcal{F}$. At the next recovery stage s' after s, $f_{e,s'}$ maps a component C_H from $\mathcal{E}_e$ to H such that C_H is the only component in $\mathcal{E}_e$ containing $\mathbf{m}^{\mathbf{H}}_\alpha$, and thus $C_H \cong H$.

– Case 2. The component H is $\mathcal{W}^{\mathrm{M}}_{\alpha,s}$ and therefore is marked by α in the next stage s'. At stage s', $f_{\alpha,s'}$ maps a component C_H in $\mathcal{E}_e$ to H. Suppose for the sake of contradiction that the component C_H is not isomorphic to H. Then C_H must be isomorphic to a component that is marked by α at a later stage. In other words, there is a stage $t' > s'$ such that $C_H \cong M_{\alpha,t'}$. At stage t', $M_{\alpha,t'}$ contains $\mathbf{m}_\alpha^{\mathbf{M}_{\alpha,t'}}$ that is unique in $\mathcal{F}_{t'}$. Let $\ell \geq t'$ be the least stage where R_e recovers at α and C_H has completely revealed itself. Every stage $i \in \{t', \ldots, \ell\}$ where R_e recovers at α the construction will define a new α-marked component $M_{\alpha,i}$ and $f_{\alpha,i}$ will map a component in $\mathcal{E}_e$ isomorphically to $M_{\alpha,i}$. Both $M_{\alpha,i}$ and $f^{-1}_{\alpha,i}(M_{\alpha,i})$ contain $\mathbf{m}_\alpha^{\mathbf{M}_{\alpha,t'}}$. Therefore at stage ℓ, the number of components in $\mathcal{E}_{e,\ell}$ containing $\mathbf{m}_\alpha^{\mathbf{M}_{\alpha,t'}}$ (all $f^{-1}_{\alpha,i}(M_{\alpha,i})$ and C_H) is one more than the number of components in $\mathcal{F}_\ell$ containing $\mathbf{m}_\alpha^{\mathbf{M}_{\alpha,t'}}$ (all $M_{\alpha,i}$). Note also that $m_\alpha^{M_{\alpha,t'}} \in \mathsf{Check}_{\alpha,\ell}$. Therefore the construction would set $\mathsf{Forbid}_\alpha = m_\alpha^{M_{\alpha,t'}}$ and guarantee that $\mathcal{E}_e \not\cong \mathcal{F}$. Contradiction. Therefore $C_H \cong H$.

We have proved that f_α is indeed a partial isomorphism that maps components in $\mathcal{E}_e$ isomorphically to $\mathcal{F}(e)$. Hence f is an isomorphism from $\mathcal{E}_e$ to $\mathcal{F}$. ∎

Lemma 6.5.12 implies that $\mathcal{F}$ is computably categorical. It remains to prove that every component in the structure $\mathcal{F}$ is finite. The next lemma concludes the proof of Proposition 6.5.5.

Lemma 6.5.13 *Each component in $\mathcal{F}$ is finite.*

Proof. Fix a component C in $\mathcal{F}$. If $C \in \mathcal{O}_{\alpha,t} \setminus \mathcal{O}^{\mathrm{M}}_{\alpha,t}$ for some α, t, then it will never be extended at any stages $t' > t$. Hence C is a finite component. Suppose for all $\alpha \in T$, $t \in \mathbb{N}$, $C \in \mathcal{O}_{\alpha,t}$ implies $C \in \mathcal{O}^{\mathrm{M}}_{\alpha,t}$. We have two cases:

– Case 1: For some e and $t > s_e$, $C \in \mathcal{O}_{\alpha,t}$ for some node $\alpha <_{\mathrm{L}} \delta \upharpoonright e$. After s_e, the construction will never act on α again and so C would not be extended any further.

– Case 2: For all e and all $t > s_e$, $C \notin \mathcal{O}_{\alpha,t}$ for any node $\alpha <_{\mathrm{L}} \delta \upharpoonright e$. For all $e \in \mathbb{N}$, if $C \in \mathcal{O}_{\delta\upharpoonright e,t}$ for some $t \in \mathbb{N}$, then $C \in \mathcal{O}^{\mathrm{M}}_{\delta\upharpoonright e,t}$. Note that for all $d > e$, the component A_d are taken from the components

$$\mathcal{M}_e = \bigcup_{s>s_e} C^{\mathrm{M}}_{\delta\upharpoonright e,s} \cup \{M_{\delta\upharpoonright e,s}\} \cup \mathcal{W}^{\mathrm{M}}_{\delta\upharpoonright e,s} \cup \mathcal{O}^{\mathrm{M}}_{\delta\upharpoonright e,s}.$$

Also note that there are only finitely many components in $\mathcal{M}_e$ that are before the component C. Therefore eventually, $\mathcal{M}_e$ will be set as A_d for some $d > e$.

In both cases, the component C is finite. This proves the lemma and hence Prop. 6.5.5 is proved. ∎

Bibliography

[1] M. Ajtai and R. Fagin. Reachability is harder for directed than for undirected graphs. *Journal of Symbolic Logic*, 55:113–150, 1990.

[2] S. Arora and R. Fagin. On winning strategies in Ehrenfeucht-Fraïssé games. *Theoretical Computer Science*, 174:97–121, 1997.

[3] V. Bárány. *Automatic Presentations of Infinite Structures*. PhD thesis, RWTH Aachen, 2007.

[4] V. Bárány, E. Grädel, and S. Rubin. Automata-based presentations of infinite structures. 2010.

[5] M. Benedikt, L. Libkin, and F. Neven. Logical definability and query languages over ranked and unranked trees. *ACM Trans. Comput. Log.*, 8(2), 2007.

[6] A. Blumensath. Automatic structures. Diploma thesis, RWTH Aachen, 1999.

[7] A. Blumensath and E. Grädel. Automatic structures. In *Procedings of the 15th Annual IEEE Symposium on Logic in Computer Science, LICS'00*, pages 51–62. IEEE Computer Society Press, 2000.

[8] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, 37(6):641–674, 2004.

[9] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of the 8th International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.

[10] J. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[11] J. R. Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Logic, Methodology, and Philosophy of Science: Proc. of the 1960 International Congress*, pages 1–11. Stanford University Press, 1962.

[12] P. Cholak, S. Goncharov, B. Khoussainov, and R. Shore. Computably categorical structures and extensions by constants. *Journal of Symbolic Logic*, 64:13–37, 1999.

[13] T. Colcombet and C. Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science*, 3(2), 2007.

[14] B. F. Csima, B. Khoussainov, and J. Liu. Computable categoricity of graphs with finite components. In *Proceedings of the 4th Conference on Computability in Europe (CiE'08)*, volume 5028 of *Lecture Notes in Computer Science*, pages 139–148. Springer, 2008.

[15] A. Dawar. Infinitary logic and inductively definability over finite structures. *Information and Computation*, 119(2):160–175, 1995.

[16] C. Delhommé. Rado's graph is not automatic. 2001. manucript.

[17] R. Downey. On presentations of algebraic structures. In *in Complexity, Logic and Recursion Theory*, pages 157–205. Dekker, 1995.

[18] V. Dzgoev and S. Goncharov. Autostability of models. *Algebra i Logika*, 19:45–58, 1980.

[19] H. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.

[20] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961.

[21] C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Am. Math. Soc.*, 98:21–51, 1961.

[22] J. L. Ershov. *Decision problems and constructive models*. Nauka, Moscow, 1980.

[23] Y. L. Ershov, S. S. Goncharov, A. Nerode, J. B. Remmel, and V. W. Marek, editors. *Handbook of recursive mathematics. Vol. 1 and Vol. 2*, volume 138-139 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1998.

[24] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 232–247. Springer, 2000.

[25] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. Karp, editor, *Compleixty of Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 43–73, 1974.

[26] R. Fagin. Monadic generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975.

[27] R. Fagin, L. Stockmeyer, and M. Vardi. On monadic NP vs monadic co-NP. *Information and Computation*, 120:78–92, 1994.

[28] R. Fraïssé. Sur quelques classifications des systeémes de relations. *Université d'Alger, Publications Scientifique, Série A*, 1:35–182, 1954.

[29] A. Fröhlich and J. C. Shepherdson. On the factorisation of polynomials in a finite number of steps. *Math. Z.*, 62:331–334, 1955.

[30] A. Fröhlich and J. C. Shepherdson. Effective procedures in field theory. *Philos. Trans. Roy. Soc. London. Ser. A.*, 248:407–432, 1956.

[31] H. Gaifman. On local and non-local properties. In *Proc. Herbrand Symp., Logic Colloquium '81*, pages 105–132. North-Holland, 1982.

[32] S. Göller and M. Lohrey. Branching-time model checking of one-counter processes. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, volume 5 of *LIPIcs*, pages 405–416. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[33] S. Göller, R. Mayr, and A. W. To. On the computational complexity of verifying one-counter processes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science (LICS'09)*, pages 235–244. IEEE Computer Society Press, 2009.

[34] S. Goncharov. Automstability of models and abelian groups. *Algebra i Logika*, 19(1):23–44, 1980.

[35] S. Goncharov and B. Khoussainov. Open problems in the theory of constructive algebraic systems. In *Computability theory and its applications*, volume 257 of *Contemp. Math.*, pages 145–170. Amer. Math. Soc., 2000.

[36] S. Goncharov, S. Lempp, and R. Solomon. The computable dimension of ordered abelian groups. *Adv. Math.*, 175(1):102–143, 2003.

[37] S. S. Goncharov. Selfstability and computable families of constructivizations. *Algebra i Logika*, 14(6):647–680, 1975.

[38] S. S. Goncharov. Constructive models of $\aleph_1$-categorical theories. *Mat. Zametki*, 23(6):885–888, 1978.

[39] S. S. Goncharov. Computable univalent numerations. *Algebra i Logika*, 19(5):507–551, 1980.

[40] S. S. Goncharov. The problem of the number of nonautoequivalent constructivizations. *Algebra i Logika*, 19(6):621–639, 1980.

[41] S. S. Goncharov. Limit equivalent constructivizations. *Mathematical logic and the theory of algorithms*, 2:4–12, 1982.

[42] E. Grädel, P. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Springer-Verlag, 2005.

[43] E. Griffor, editor. *Handbook of computability theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1999.

[44] M. Grohe. Equivalence in finite-variable logics is complete for polynomial time. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, pages 264–273. Society Press, 1996.

[45] S. Grumbach and J. Su. Finitely representable databases. *J. Comput. Syst. Sci.*, 55(2):273–298, 1997.

[46] Y. Gurevich. Toward logic tailored for computer science. In R. et al., editor, *Computation and proof theory*, volume 1104 of *Lecture Notes in Mathematics*, pages 175–216. Springer, 1984.

[47] P. Hájek. Generalized quantifiers and finite sets. *Prace Nauk. Inst. Mat. Politech. Wroclaw No. 14 Ser. Konfer*, (1):91–104, 1977.

[48] W. Hanf. Model-theoretic methods in the study of elementary logic. In J. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*, pages 132–145. North-Holland, 1965.

[49] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[50] L. Hella. Logical hierarchies in ptime. *Information and Computation*, 129:1–19, 1996.

[51] L. Hella, L. Libkin, and J. Nurmonen. Notions of locality and their logical characterizations over finite models. *Journal of Symbolic Logic*, 64:1751–1773, 1999.

[52] D. R. Hirschfeldt. Degree spectra of relations on computable structures. *Bulletin of Symbolic Logic*, 6(2):197–212, 2000.

[53] G. Hjorth, B. Khoussainov, A. Montalbán, and A. Nies. From automatic structures to borel structures. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, Pittsburgh, PA, USA*, pages 431–441, 2008.

[54] W. Hodges. *Model theory*. Encyclopedia of Mathematics. Cambridge University Press, 1993.

[55] B. R. Hodgson. On direct products of automaton decidable theories. *Theoretical Computer Science*, 19:331–335, 1982.

[56] J. Honkala. On the problem whether the image of an N-rational series equals N. *Fund. Inform.*, 73(1-2):127–132, 2006.

[57] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison Wesley, 2000.

[58] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of Sixth Annual ACM Symposium on Theory of Computing (STOC'74)*, pages 172–184, 1974.

[59] N. Immerman. Ralational queries computable in polynomial time (extended abstract). In *ACM Symp. on Theory of Computing*, pages 147–152. ACM Press, 1982.

[60] N. Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25:76–98, 1982.

[61] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[62] L. Kaiser, S. Rubin, and V. Bárány. Cardinality and counting quantifiers on omega-automatic structures. In *Procedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science, STACS'08, Bordeaux, France*, pages 385–396, 2008.

[63] C. Karp. Finite quantifier equivalence. In J. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*, pages 407–412. North Holland, 1965.

[64] B. Khoussainov and J. Liu. On complexity of Ehrenfeucht-Fraïssé games. In *Proceedings of International Symposium on Logical Foundations of Computer Science (LFCS'07)*, volume 4514 of *Lecture Notes in Computer Science*, pages 293–309. Springer, 2007.

[65] B. Khoussainov and J. Liu. On complxity of Ehrenfeucht-Fraïssé games. *Annals of Pure and Applied Logic*, 161(3):404–415, 2009.

[66] B. Khoussainov, J. Liu, and M. Minnes. Unary automatic graphs: An algorithmic perspective. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC'08)*, volume 4978 of *Lecture Notes in Computer Science*, pages 542–553. Springer, 2008.

[67] B. Khoussainov, J. Liu, and M. Minnes. Unary automatic graphs: an algorithmic perspective. *Mathematical Structures in Computer Science*, 19(1):133–152, 2009.

[68] B. Khoussainov and M. Minnes. Three lectures on automatic structures. In *Proceedings of the Logic Colloquium '07*, volume 35 of *Lecture Notes in Logic*, pages 132–176–704. Springer, 2007.

[69] B. Khoussainov and M. Minnes. Model theoretic complexity of automatic structures. In *Proceedings of TAMC 08*. Springer, 2008. to appear.

[70] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC: International Workshop on Logic and Computational Complexity*, number 960 in LNCS, pages 367–392, 1995.

[71] B. Khoussainov and A. Nerode. Open questions in the theory of automatic structures. *Bulletin of the EATCS*, 94:184–204, 2008.

[72] B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: Richness and limitations. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS'04), Turku (Finnland)*. IEEE Computer Society Press, 2004. 44–53.

[73] B. Khoussainov and S. Rubin. Graphs with automatic presentations over a unary alphabet. *Journal of Automata, Languages and Combinatorics*, 6(4):467–480, 2001.

[74] B. Khoussainov and S. Rubin. Graphs with automatic presentations over a unary alphabet. *Journal of Automata, Lnaguages and Combinatorics*, 6(4):467–480, 2001.

[75] B. Khoussainov, S. Rubin, and F. Stephan. Automatic partial orders. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 168–177. IEEE Computer Society Press, 2003.

[76] B. Khoussainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In V. Diekert and M. Habib, editors, *Proceedings of the 21th Annual Symposium on Theoretical Aspects of Computer Science (STACS'04), Montpellier (France)*, number 2996 in LNCS, pages 440–451. Springer, 2004.

[77] B. Khoussainov, S. Rubin, and F. Stephan. Automatic linear orders and trees. *ACM Transactions on Computational Logic*, 6(4):675–700, 2005.

[78] B. Khoussainov and R. Shore. Computable isomorphisms, degree spectra of relations, and scott families. *Annals of Pure and Applied Logic*, 93(1-3):153–193, 1998.

[79] B. Khoussainov and R. A. Shore. Effective model theory: the number of models and their complexity. In *Models and computability (Leeds, 1997)*, volume 259 of *London Math. Soc. Lecture Note Ser.*, pages 193–239. Cabridge Univ. Press, 1999.

[80] P. Kolaitis and J. Panttaja. On the complexity of existential pebble games. In *Proceedings of the 17th International Workshop on Computer Science Logic (CSL'03)*, volume 2803 of *LNCS*, pages 314–329, 1999.

[81] D. Kuske, J. Liu, and M. Lohrey. The isomorphism problem on classes of automatic structures. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS'10)*. IEEE Computer Society, 2010.

[82] D. Kuske, J. Liu, and M. Lohrey. The isomorphism problem on classes of automatic structures. *CoRR*, abs/1001.2086, 2010.

[83] D. Kuske and M. Lohrey. First-order and counting theories of *mega*-automatic structures. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'06)*, pages 322–336, 2006.

[84] S. Lempp, C. McCoy, R. Miller, and R. Solomon. Computable categoricity of trees of finite height. *J. Symbolic Logic*, 70(1):151–215, 2005.

[85] L. Libkin. *Elements of Finite Model Theory*. Springer-Verlag, 2004.

[86] J. Liu and M. Minnes. Analysing complexity in classes of unary automatic structures. In *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications (LATA'09)*, volume 5457 of *Lecture Notes in Computer Science*, pages 518–529. Springer, 2009.

[87] A. I. Mal'cev. Constructive algebras. *I. Uspehi Mat. Nauk*, 16(3(99)):3–60, 1961.

[88] Y. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993.

[89] G. Metakides and A. Nerode. Recursion theory and algebra. *Algebra and logic (Fourteenth Summer Res. Inst., Austral. Math. Soc., Monash Univ., Clayton, 1974)*, 450, 1975.

[90] T. Millar. Foundations of recursive model theory. *Ann. Math. Logic*, 13(1):45–72, 1978.

[91] M. Minnes. *Computability and Complexity Properties of Automatic Structures and their Applications*. PhD thesis, Cornell University, 2008.

[92] A. Nies. Describing groups. *Bull. Symbolic Logic*, 13(3):305–339, 2007.

[93] G. P. Oliver and R. M. Thomas. Automatic presentations for finitely generated groups. In V. Diekert and B. Durand, editors, *Proceedings of the 22th Annual Symposium on Theoretical Aspects of Computer Science (STACS'05)*, number 3404 in LNCS, pages 693–704. SPRINGER, 2005.

[94] E. Pezzoili. Computational complexity of ef games on finite structures. In *Proceedings of the 12th International Workshop on Computer Science Logic (CSL'98)*, pages 159–170, 1999.

[95] B. Poizat. Deux ou trois choses que je sais de l_n. *Journal of Symbolic Logic*, 47:641–658, 1982.

[96] M. O. Rabin. Recursive unsolvability of group theoretic problems. *Ann. of Math.*, 67(2):172–194, 1958.

[97] M. O. Rabin. Computable algebra, general theory and theory of computable fields. *Trans. Amer. Math. Soc.*, 95:341–360, 1960.

[98] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[99] J. Remmel. Recursively categorical linear orderings. *Proce. Amer. Math. Soc.*, pages 387–391, 1981.

[100] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1968.

[101] J. Rosenstein. *Linear Ordering*. Academic Press, 1982.

[102] S. Rubin. *Automatic Structures*. PhD thesis, University of Auckland, 2004.

[103] S. Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14:169–209, 2008.

[104] K. Salomaa, C. Campeanu, K. C. II, and S. Yu. State complexity of basic operations on finite languages. In *Proceedings of Fourth International Workshop on Implementing Automata (WIA'99)*, volume 2214 of *Lecture Notes in Computer Science*, pages 60–70. Springer, 2001.

[105] T. Schwentick. On winning Ehrenfeucht games and monadic np. *Annals of Pure and Applied Logic*, 79:61–92, 1996.

[106] D. Scott. Logic with denumerable long formulas and finite strings of quantifiers. In A. T. L. Henkin, editor, *The Theory of Models*, volume 1104, pages 329–341. North Holland, 1965.

[107] O. Serre. Parity games played on transition graphs of one-counter processes. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'06)*, volume 3921 of *LNCS*, pages 337–351. Springer, 2006.

[108] R. Soare. Tree arguments in recursion theory and the 0'''-priority method. In *Recursion Theory, Proc. Symp. Pure Math. 42*, pages 53–106. Amer. Math. Soc., 1985.

[109] R. Soare. *Recursively enumerable sets and degrees*. Perspectives in Mathematical Logic. Springer-Verlag, 1987.

[110] W. Tait. A counterexample to a conjecture of scott and suppes. *Journal of Symbolic Logic*, 24:15–16, 1959.

[111] W. Thomas. A short introduction to infinite automata. In *Proceedings of the 5th International Conference on Developments in Language Theory (DLT'01)*, volume 2295 of *LNCS*, pages 130–144. Springer, 2001.

[112] A. W. To. Model checking FO(r) over one-counter processes and beyond. In *Proceedings of the 23rd international Workshop on Computer Science Logic (CSL'09)*, volume 5771 of *LNCS*, pages 485–499. Springer, 2009.

[113] D. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Doklady Academii Nauk SSSR*, 70:569–572, 1950.

[114] T. Tsankov. The additive group of the rationals does not have an automatic presentation. preprint, 2009.

[115] B. L. van der Waerden. Eine bemerkung über die unzerlegbarkeit von polynomen. *Mathematische Annalen*, 102:738–739, 1930.

[116] M. Vardi. The complexity of relational query languages. In *ACM Symp. on Theory of Computing*, pages 137–146. ACM Press, 1982.

[117] M. Vardi. Model checking for database theoreticians. In *Proc. 10th International Conference on Database Theory*, pages 1–16, 2005.

[118] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *In Proceedings of the Symposium on Logic in Computer Science (LICS'86)*, pages 332–344, 1986.

[119] S. Yu. Regular languages. In *Handbook of Formal Languages*, pages 40–110. Springer, 1997.

[120] S. Yu. State complexity: recent results and open problems. *Fundamenta Informaticae*, 64(1-4):471–480, 2005.

List of Notations

— Chapter 5 —
Σ_1^1, 97
$\mathcal{A}_1 \uplus \mathcal{A}_2$, 98
$D \uplus \mathcal{A}_2$, 99
$\mathcal{A}[p(\overline{x})]$, 99
$\mathcal{A}_1 \times \mathcal{A}_2$, 99
$\otimes_k(L)$, 99
$\mathrm{Run}_{\mathcal{A}} = (S, I, \Delta', F)$, 100
$C(x, y)$, 100
$H_1 \sim H_2$, 105
H_1^{ω}, 105
$r \circ H$, 105
unfold(D), 110
$L_1 + L_2$, 115
$L_1 \cdot L_2$, 115
$\sum \mathcal{L}$, 115
$\leq_{\mathrm{lex}}$, 116
$L[n_1, n_2]$, 117
Shuf$(\mathcal{L})$, 117
$\sqsubseteq$, 122
$\sigma(D)$, 123
$D\mathcal{A}$, 125
$\mathcal{A}[q_1, q_2]$, 126

— Chapter 6 —
dim$(\mathcal{S})$, 133
$\mathfrak{G}_{\mathsf{SLF}}$, 135
$\mathcal{C}_n$, 135
$\mathcal{S}_n$, 135
$\mathcal{L}_n$, 135
$\mathcal{C}'_n$, 136
$\mathcal{C}''_n$, 136
$C(v)$, 136
size$_{\mathcal{G}}$, 136
ext$_{\mathcal{G}}$, 137
$\mathbf{n}$, 146
$\mathcal{E}_i$, 147
$\leq_{\mathrm{L}}$, 148
$\leq_{\mathrm{pref}}$, 148
$A_{\alpha,t}$, 149
$M_{\alpha,t}$, 149
$\mathcal{C}^{\mathrm{M}}_{\alpha,t}$, 149
$\mathcal{C}_{\alpha,t}$, 149
$\mathcal{O}^{\mathrm{M}}_{\alpha,t}$, 149
$\mathcal{O}_{\alpha,t}$, 149
$\mathcal{W}^{\mathrm{M}}_{\alpha,t}$, 149
$\mathcal{W}_{\alpha,t}$, 149
$\delta(n)$, 149
$\delta \restriction n$, 149
$\mathcal{F}_{\alpha,t}$, 150

Index

Name Index