COMPUTABILITY AND COMPLEXITY PROPERTIES OF AUTOMATIC STRUCTURES AND THEIR APPLICATIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Mor Mia Minnes

August 2008

COMPUTABILITY AND COMPLEXITY PROPERTIES OF AUTOMATIC STRUCTURES AND THEIR APPLICATIONS

Mor Mia Minnes, Ph.D.

Cornell University 2008

Finite state automata are Turing machines with fixed finite bounds on resource use. Automata lend themselves well to real-time computations and efficient algorithms. Continuing a tradition of studying computability in mathematics, we examine **automatic structures**, mathematical objects which can be represented by automata, and apply resulting observations to computer science.

We measure the complexity of automatic structures via well-established concepts from model theory, topology, and set theory. We prove the following results. The ordinal height of any automatic well-founded partial order is bounded by ω^ω. The ordinal heights of automatic well-founded relations are unbounded below ω_1^{CK}, the first uncomputable ordinal. For any computable ordinal α, there is an automatic structure of Scott rank at least α. Moreover, there are automatic structures of Scott rank $\omega_1^{CK}, \omega_1^{CK} + 1$. For any computable ordinal α, there is an automatic successor tree of Cantor-Bendixson rank α.

Next, we study infinite graphs produced from a natural unfolding operation applied to finite graphs. Graphs produced via such operations have finite degree and can be described by finite automata over a one-letter alphabet. We investigate algorithmic properties of such graphs in terms of their finite presentations. In particular, we ask how hard it is to check whether a given node belongs to an infinite component, whether two given nodes in the graph are reachable from one another, and whether the graph is connected. We give polynomial-time algorithms

answering each of these questions. For a fixed input graph, the algorithm for infinite component membership works in constant time and reachability is decided uniformly by a single automaton. Hence, we improve on previous work, in which nonelementary or nonuniform algorithms were found.

We turn our attention to automata techniques for deciding first-order logical theories. These techniques are useful in Integer Linear Programming and Mixed Integer Linear Programming, which in turn have applications in diverse areas of computer science and engineering. We extend known work to address the enumeration problem for linear programming solutions. Then, we apply a similar paradigm to give an automata theoretic decision procedure for the p-adic valued ring under addition and for formal Laurent series over a finite field with valuation and addition.

BIOGRAPHICAL SKETCH

Mor Minnes was born in Haifa, Israel in 1982. From an early age, Mor learned the importance of words and education: in the bath, Mor's parents (both students at the time) saw her and her sister typing intently on imaginary typewriters and proclaiming that they were working. In 1989, the nuclear family (now also including two brothers) moved to Vancouver, Canada for what was intended as a short visit for Mor's father to do a residency in psychiatry. Quickly, Mor became Mia, choosing a more suitable English name at the suggestion of her maternal grandmother. The family settled in Vancouver and Mia attended Prince of Wales Mini School and the International Baccalaureate Program at Sir Winston Churchill Secondary School. While in high school, Mia enjoyed both the sciences and the humanities. Not willing to give up either of these, Mia decided to pursue dual Bachelor's degrees at Queen's University in Kingston, Ontario. In 1999, Mia headed to snowy Kingston and enrolled in Applied Science (Mathematics & Engineering, Computing and Communications) and Philosophy. Mia had her first encounter with programming and fell in love with the power behind logical thinking. Considering graduate studies, logic was a natural fit with Mia's interests as it lay in the intersection of her favourite philosophy courses (philosophy of language and science) and engineering courses (computer architecture, programming, and math). Her best friend compiled a list of graduate programs in logic and Mia started browsing through their websites. When she discovered Anil Nerode's research outlining connections between automata, logic, and hybrid systems, she knew she had found what she wanted to work on. Mia graduated from Queen's and headed south to Ithaca, NY to begin graduate school in mathematics at Cornell University, along the way also earning a Master's degree in computer science. After five years of graduate school, Mia is looking forward to being a C.L.E. Moore Instructor at MIT next year.

ACKNOWLEDGEMENTS

First, I thank my advisor, Anil Nerode. Anil personifies the possibilities and excitement of integrating theoretical and applied research. He has been an inspiring source of ideas, historical context, and anecdotes. Moreover, he has guided me through my initial forays into the world of academic mathematics and has been an incredible advocate in facilitating interactions with leading researchers. In particular, Anil introduced me to Bakhadyr Khoussainov. I am grateful to Bakh for accepting me as a colleague and for introducing me to automatic structures. He also invited me to Auckland, New Zealand and hosted a productive and highly enjoyable month of work and travel.

My committee members Richard Shore, Dexter Kozen, and Joseph Halpern have been instrumental in bringing me to this point in my academic career. The courses I took from each of them during my first years in graduate school helped shape my view of mathematical logic and theoretical computer science. Through his leadership of the logic seminar and his incisive questions, Richard has helped me hone my academic speaking and writing skills and find clarity in complicated arguments. Also, in our conversations during frequent commutes between Ithaca and the Boston area, Richard gave me invaluable professional advice and glimpses at his brilliant grasp of the big picture of our field. Dexter and Joe illustrated future directions and connections of my research within computer science and helpfully pointed me to references which broadened my understanding of the material.

I have greatly enjoyed speaking with and learning from many researchers. The New Zealand group, including André Nies, Rod Downey, Noam Greenberg, Sasha Rubin, and Jiamou Liu, provided a stimulating work environment during my visit and at conferences and talks over the past five years. Along the way, I also benefited from mathematical conversations with Doug Cenzer, Peter Cholak, Barbara Csima,

Joe Miller, Antonio Montalbán, Reed Solomon, Frank Stephan, and Moshe Vardi. My fellow graduate students in the mathematics department (in particular, my peers in the logic group and my friends from the 120A Café) at Cornell have made these past years immeasurably richer and more fun through our philosophical and mathematical discussions. In her capacity as the graduate student teaching coordinator, and as a personal friend, Maria Terrell has supported me during my graduate career. Her flexibility has given me challenging and enriching teaching opportunities in some semesters, while allowing me the freedom to go to conferences and research visits in others.

Last, but certainly not least, I would like to thank my family. My mother has been a pillar of strength, insight, and inspiration. For as long as I can remember, she has made juggling two or three careers and taking care of the family look easy and rewarding. She made me believe I could do anything I want. My father, with his sense of humour and his patience, reminded me that it's okay to make mistakes and that if you listen to yourself, you find the path that's right for you. I can't imagine growing up without my siblings: Shir, thank you for listening to my rants and my excitement - it means so much to me that we are so close; Tom, we've been the meat and veggies of our crazy family sandwich and you make us all laugh so much; Gal, thank you for always wanting to learn about the math that I do even though I know it's so remote from your interests. I have been inspired by my parents-in-law's pride in me to live up to their expectations. And, Todd – your support, encouragement, and love have nourished me along this journey. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction and Preliminaries

1.1 Motivation and background

In recent years there has been increasing interest in the study of structures that can be presented by automata. The underlying idea in this line of research consists of using automata (such as finite automata, Büchi automata, tree automata, and Rabin automata) to represent structures and study the logical and algorithmic consequences of such presentations. Informally, a structure $\mathcal{A} = (A; R_0, \ldots, R_m)$ is **automatic** if the domain A and all the relations R_0, ..., R_m of the structure are recognised by finite automata (precise definitions are in Section 1.2). For instance, an automatic graph is one whose set of vertices and set of edges can each be recognised by finite automata. This definition is analogous to that of computable structures, in which the domain and all basic relations are required to be computable.

In the 1980s, as part of their feasible mathematics program, Nerode and Remmel [78] suggested the study of polynomial-time structures. A structure is said to be polynomial-time if its domain and relations can be recognised by Turing machines that run in polynomial time. An important early result by Cenzer and Remmel [27] showed that every computable purely relational structure is computably isomorphic to a polynomial-time structure. This implies that solving questions about the class of polynomial-time structures is as hard as solving them for the class of computable structures. For instance, the problem of classifying the isomorphism types of polynomial-time structures is as hard as that of classifying the

isomorphism types of computable structures. Since polynomial-time structures and computable structures yielded similar complexity results, greater restrictions on models of computations were imposed. In 1995, Khoussainov and Nerode suggested bringing in models of computations that have less computational power than polynomial-time Turing machines. The hope was that if these weaker machines were used to represent the domain and basic relations, then perhaps isomorphism invariants could be more easily understood. Specifically, they suggested the use of finite state machines (automata) as the basic computation model.

The idea of using automata to study structures goes back to the work of Büchi. Büchi [22], [23] used automata to prove the decidability of of a theory called $S1S$ (monadic second-order theory of the natural numbers with one successor). Rabin [85] then used automata to prove that the monadic second-order theory of two successor functions, $S2S$, is also decidable. In the realm of logic, these results have been used to prove decidability of first-order or MSO theories. Büchi knew that automata and Presburger arithmetic (the first-order theory of the natural numbers with addition) are closely connected. He used automata to give a simple proof (not using quantifier elimination) of the decidability of Presburger arithmetic. Capturing this notion, Hodgson [49] defined automaton decidable theories in 1982. While he coined the definition of automatic structures, little was done in the 1980s to follow up on his work. In 1995, Khoussainov and Nerode [56] rediscovered the concept of automatic structure and initiated a systematic study of the area.

Automatic structures possess a number of nice algorithmic and model-theoretic properties. For example, Khoussainov and Nerode proved that the first-order theory of any automatic structure is decidable [56]. This result is extended by adding the $\exists^{\infty}$ (there are infinitely many) and $\exists^{n,m}$ (there are n many mod m) quantifiers

to the first-order logic [14],[62]. Blumensath and Grädel proved a logical characterization theorem stating that automatic structures are exactly those definable in a particular fragment of arithmetic (see Example 1.6). Automatic structures are closed under first-order interpretations. There are descriptions of automatic linear orders and trees in terms of model theoretic concepts such as Cantor-Bendixson ranks [63]. Also, Khoussainov, Nies, Rubin and Stephan have characterized the isomorphism types of automatic Boolean algebras [59]; Thomas and Oliver have given a full description of finitely generated automatic groups [82] and a recent result by Nies and Thomas [81] gives a necessary condition for infinite groups to be automatic. Some of these results have direct algorithmic implications. For example, the isomorphism problems for automatic well-ordered sets and Boolean algebras are decidable [59].

Thurston observed that many finitely generated groups associated with 3-manifolds are finitely presented groups with the property that finite automata recognise equality of words and the graphs of the unary operations of left multiplication by a generator; these are the Thurston automatic groups. These groups yield rapid algorithms [38] for computing topological and algebraic properties of interest (such as the word problem). Among these groups are Coxeter groups, braid groups, Euclidean groups, and others. We emphasize that Thurston automatic groups differ from automatic groups in our sense; in particular, the vocabulary of the associated structures is starkly different. Thurston automatic groups are represented as unary algebras whose relations are all unary operations (corresponding to left multiplication by each generator). On the other hand, an automatic group in our sense represents the full group multiplication (a binary function) and hence must satisfy the constraint that the graph of this operation be recognisable by a finite automaton. The Thurston requirement for automaticity applies only to

finitely generated groups but includes a wider class of finitely generated groups than what we call automatic groups. For example, the countable direct product of $(\mathbb{Z}; +)$ is an automatic structure (see Example 1.12 and Lemma 1.10) but is not a Thurston automatic group because it is not finitely generated. Free groups on two or more generators are Thurston automatic but not automatic in our sense because their full multiplication is provably not recognisable by any finite automaton.

In the computer science community, an interest in automatic structures comes from problems related to model checking. Model checking is motivated by the quest to prove correctness of computer programs. This subject allows infinite state automata as well as finite state automata. Current topics of interest may be found in [2], [1], [19]. Examples of infinite state automata include concurrency protocols involving an arbitrary number of processes, programs manipulating some infinite sets of data (such as the integers or reals), pushdown automata, counter automata, timed automata, Petri-nets, and rewriting systems. Given such an automaton and a specification (formula) in a formal system, the model checking problem asks us to compute all the states of the system that satisfy the specification. Since the state space is infinite, the process of checking the specification may not terminate. Specialized methods are needed to cover even the problems encountered in practice. Abstraction methods try to represent the behaviour of the system in finite form. Model checking then reduces to checking a finite representation of the state space to identify the states that satisfy the specification. Automatic structures arise naturally in infinite state model checking since both the state space and the transitions of infinite state systems are usually recognisable by finite automata. In 2000, Blumensath and Grädel [13] studied definability problems for automatic structures and the computational complexity of model checking for automatic structures.

There is also a body of work devoted to the study of resource-bounded complexity of the first-order theories of automatic structures. For example, on the one hand, Grädel and Blumensath constructed automatic structures whose first-order theories are nonelementary [14]. On the other hand, Lohrey in [71] proved that the first-order theory of any automatic graph of bounded degree is elementary. It is worth noting that when both a first-order formula and an automatic structure $\mathcal{A}$ are fixed, determining if a tuple $\bar{a}$ from $\mathcal{A}$ satisfies $\varphi(\bar{x})$ can be done in linear time.

The results about automatic structures can be seen to pull in two opposite directions. One body of work about automatic structures demonstrates that in various concrete senses automatic structures are not complex from a logical point of view. Such papers include [5], [13], [34], [52], [60], [61], [63], [80]. However, this intuition can be misleading. For example, in [59] it is shown that the isomorphism problem for automatic structures is Σ^1_1-complete. This informally tells us that there is no hope for a description (in a natural logical language) of the isomorphism types of automatic structures. A group of papers including [53], [59], [69] gives further evidence to the richness of automatic structures. There has been a series of PhD theses in the area of automatic structures including Blumensath [11], Rubin [90], Bárány [6], this thesis, and the upcoming [70]. A recently published paper of Khoussainov and Nerode [58] discusses open questions in the study of automatic structures. There are also survey papers on some of the areas in the subject by Khoussainov and Minnes [54], Nies [79], and Rubin [91].

1.2 Automatic structures and computable structures

To establish notation, we briefly recall some definitions associated with finite automata. A **finite automaton** $\mathcal{M}$ over an alphabet Σ is a tuple (S, ι, Δ, F), where S is a finite set of **states**, $\iota \in S$ is the **initial state**, $\Delta \subset S \times \Sigma \times S$ is the **transition relation**, and $F \subset S$ is the set of **final** or **accepting states**. The set of words of finite length over Σ is denoted Σ^*. A **computation** of $\mathcal{A}$ on a word $\sigma_1\sigma_2\ldots\sigma_n$ ($\sigma_i \in \Sigma$) is a sequence of states $q_0, q_1, \ldots, q_n$ such that $q_0 = \iota$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all $i \in \{0, \ldots, n-1\}$. If $q_n \in F$ the computation is **successful** and we say that the automaton $\mathcal{M}$ **accepts** the word $\sigma_1\sigma_2\ldots\sigma_n$ if there is some successful computation of $\mathcal{M}$ on it. The **language** accepted by the automaton $\mathcal{M}$ is the set of all words accepted by $\mathcal{M}$. In general, $D \subset \Sigma^*$ is **finite automaton recognisable**, or **regular**, if D is the language accepted by some finite automaton $\mathcal{M}$. An automaton (S, ι, Δ, F) is called **deterministic** if Δ is a function; that is, for each pair $(s, \sigma) \in S \times \Sigma$ there is at most one $s' \in S$ such that $(s, \sigma, s') \in \Delta$. Any language accepted by a non-deterministic finite automaton can also be recognised by some deterministic automaton. However, the deterministic automaton may have exponentially more states than its equivalent non-deterministic automaton. For proofs of these basic facts about finite automata, see for example [57].

To define automaton recognisable relations, we use n-variable (or n-tape) synchronous automata. An **n-tape synchronous automaton** can be thought of as a one-way Turing machine with n input tapes [37]. Each tape is semi-infinite, having written on it a word over the alphabet Σ followed by an infinite succession of blanks (denoted by $\diamond$ symbols). The automaton starts in the initial state, reads simultaneously the first symbol of each tape, changes state, reads simultaneously

the second symbol of each tape, changes state, etc., until it reads a blank on each tape. The automaton then stops and accepts the n-tuple of words if and only if it is in a final state. The set of all n-tuples accepted by the automaton is the relation recognised by the automaton. Formally, an n-tape automaton on Σ is a finite automaton over the alphabet $(\Sigma_\diamond)^n$, where $\Sigma_\diamond = \Sigma \cup \{\diamond\}$ and $\diamond \notin \Sigma$. The **convolution** of a tuple $(w_1, \ldots, w_n) \in \Sigma^{*n}$ is the string $c(w_1, \ldots, w_n)$ of length $\max_i |w_i|$ over the alphabet $(\Sigma_\diamond)^n$ which is defined as follows. Its k^{th} symbol is $(\sigma_1, \ldots, \sigma_n)$ where σ_i is the k^{th} symbol of w_i if $k \leq |w_i|$ and $\diamond$ otherwise. The **convolution of a relation** $R \subset \Sigma^{*n}$ is the language $c(R) \subset (\Sigma_\diamond)^{n*}$ formed as the set of convolutions of all the tuples in R. An n-ary relation $R \subset \Sigma^{*n}$ is **finite automaton recognisable**, or **regular**, if its convolution $c(R)$ is recognisable by an n-tape automaton.

In Chapter 4, we will consider automata whose inputs encode real numbers. To do so, we need automata which process inputs of infinite length. A **Büchi automaton** over a finite alphabet Σ is $\mathcal{M} = (S, \iota, \Delta, F)$ interpreted as in the case of finite automata. Inputs to $\mathcal{M}$ are infinite words $\alpha \in \Sigma^\omega$. A **computation** of $\mathcal{M}$ on input α is an infinite sequence of states $s_0, s_1, s_2 \ldots$ such that $s_0 = \iota$ and for each i, $(s_i, \sigma_\mathbf{i}, s_{i+1}) \in \Delta$. A computation of $\mathcal{M}$ is successful if it enters F infinitely many times; α is **accepted** by $\mathcal{M}$ if there is some successful computation of $\mathcal{M}$ on α. The language of $\mathcal{M}$, $L(\mathcal{M}) \subset \Sigma^\omega$, is the set of infinite words accepted by $\mathcal{M}$. As in the case of finite automata, we can define n-tape synchronous Büchi automata and thus get a notion of Büchi recognisability for n-ary relations on infinite words. A Büchi automaton is called **deterministic** if Δ is a (possibly partial) function. Unlike the case of automata on finite words, deterministic Büchi automata are *less* expressive than non-deterministic Büchi automata. That is, given a non-deterministic Büchi automaton, there is not necessarily a deterministic

Büchi automaton which accepts the same set of infinite words. For example, the set of infinite binary words $\{\alpha : \alpha \text{ contains only finitely many } 0s\} = \{0,1\}^* \cdot \{1\}^\omega$ can be recognised by a non-deterministic Büchi automaton but cannot be recognised by any deterministic Büchi automaton (see [57]).

Sets recognisable by finite or Büchi automata have strong closure properties. Standard product constructions show that the union or intersection of two recognisable sets is again recognisable. The projection operation on an n-ary relation R is defined to be

$$\exists x_i R = \{(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n) : \exists a \in \Sigma^*(a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_n) \in R\}$$

Looking at the definition of n-tape automata, it is easy to see that the projection of a recognisable relation is itself a recognisable relation. We now turn our attention to the complementation operation. For finite automata, the proof of closure under complementation is immediate: given a finite automaton $\mathcal{A}$, to construct the complement automaton we determinize $\mathcal{A}$ and then switch all accepting and non-accepting states. The proof that Büchi automatic sets are closed under complementation constituted one of Büchi's main early achievements [22], and the study of complementation algorithms for Büchi recognisable languages is still an active area of research.

The emptiness question for a given (finite or Büchi) automaton asks whether the set of words accepted by the automaton is empty. In each case, we have an efficient algorithm to answer this question. Given a finite automaton, its language is non-empty just in case there is a path in the underlying directed graph which begins at the initial state and ends at some accepting state. The existence of such a graph can be checked in linear time in the size of the automaton by a breadth-first search. On the other hand, non-emptiness of a Büchi automaton corresponds to

there being a lasso: a path in the underlying directed graph which begins at the initial state, reaches an accepting state, and then loops back to the accepting state (see Figure 1.1). Checking for such a lasso can also be done efficiently.

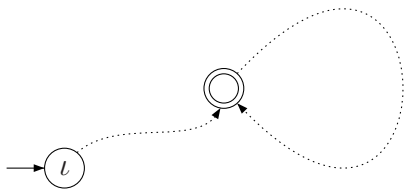

Figure 1.1: A lasso in a Büchi automaton.

We will be using automata to represent mathematical objects. Therefore, we now present the logical abstractions for discussing such objects. A **vocabulary** or **signature** is a finite sequence $(R_1^{m_1}, \ldots, R_t^{m_t}, f_1^{n_1}, \ldots, f_r^{n_r}, c_1, \ldots, c_s)$, where each $R_j^{m_j}$ is a relation (predicate) symbol of arity $m_j > 0$, each $f_i^{n_i}$ is a function symbol of arity $n_i > 0$, and each c_k is a constant symbol. We will restrict our attention to relational vocabularies, those without function or constant symbols, by representing functions by their graphs and replacing constants by unary predicates which hold of a unique element. The relations R_j of the signature are called basic or atomic relations. A **structure** (or model) of the vocabulary $(R_1^{m_1}, \ldots, R_t^{m_t})$ is a tuple $\mathcal{A} = (A; R_1^{\mathcal{A}}, \ldots, R_t^{\mathcal{A}})$, where A is the domain (or universe) of $\mathcal{A}$ and $R_j^{\mathcal{A}} \subset A^{m_j}$ is a relation interpreting the symbol R_j of the vocabulary. When convenient, we may omit the superscripts $\mathcal{A}$ of the interpretations. The cardinality of the structure is the cardinality of its domain. We only consider infinite structures, those whose universe is an infinite set, since any finite structure is recognisable by a finite automaton.

Definition 1.1. A structure $\mathcal{A} = (A; R_0, R_1, \ldots, R_m)$ is **automatic** over Σ if its domain A and all basic relations R_0, R_1, …, R_m are regular over Σ. If $\mathcal{B}$ is isomorphic to an automatic structure $\mathcal{A}$ then we call $\mathcal{A}$ an **automata presentation**

of $\mathcal{B}$ and say that $\mathcal{B}$ is **automatically presentable**.

There is a wide literature studying structures which can be presented by other types of automata (including the Büchi automata discussed above, as well as automata whose inputs are trees rather than strings). An overview of this literature is presented in the Khoussainov and Minnes survey paper [54]. This thesis contains results mainly on automatic structures where the automata are finite automata (over finite strings). We now present some examples of such structures. Our first examples are automatic structures over the alphabet $\Sigma = \{1\}$. These structures are called **unary automatic** and are discussed further in Chapter 3.

Example 1.2. The structure $(1^*; \leq, S)$, where $1^m \leq 1^n \iff m \leq n$ and $S(1^n) = 1^{n+1}$, is automatic.

Example 1.3. The structure $(1^*; \mod_1, \mod_2, \ldots, \mod_n)$, where n is a fixed positive integer, is automatic. The finite automata recognising the modular relations contain cycles of appropriate lengths.

Next, we move to structures with a binary alphabet $\{0, 1\}$. Any automatic structure over a finite alphabet is isomorphic to an automatic structure over a binary alphabet [90]. Clearly, any automatic structure has a countable domain.

Example 1.4. The structure $(\{0,1\}^*; \vee, \wedge, \neg)$ is automatic because bit-wise operations on binary strings can be recognised by finite automata.

Example 1.5. The structure $(\{0,1\}^* \cdot 1; +_2, \leq)$, where $+_2$ is binary addition if the binary strings are interpreted as base-2 encodings of natural numbers with the least significant bit first. The usual algorithm for adding binary numbers involves a single carry bit, and therefore a small finite automaton can recognise the relation $+_2$, as in Figure 1.2.

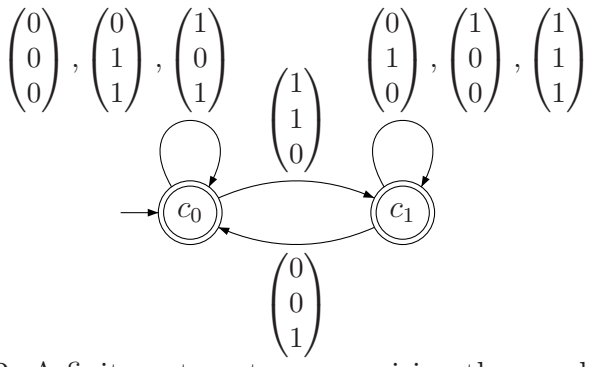

Figure 1.2: A finite automaton recognising the graph of $+_2$

Example 1.6. Instead of Presburger arithmetic, we may consider the structure $(\{0,1\}^* \cdot 1; +_2, |_2)$. This is arithmetic with weak divisibility: $w \mid_2 v$ if w represents a power of 2 which divides the number represented by v. Since we encode natural numbers by their binary representation, weak divisibility is a regular relation.

Example 1.7. If we treat binary strings at face value rather than as representations of natural numbers, we arrive at a different automatic structure:

$$(\{0,1\}^*; \preceq, Left, Right, EqL),$$

where $\preceq$ is the prefix relation, $Left$ and $Right$ denote the functions which append a 0 or 1 to the binary string (respectively), and EqL is the equal length relation. It is easy to show that this structure is automatic. This structure plays a major role in the study of automatic structures with respect to logical definability [13].

Example 1.8. A useful example of an automatic structure is the configuration space of a Turing machine. The configuration space is a graph whose nodes are the configurations of the machine (the state, the contents of the tape, and the position of the read/write head). An edge exists between two nodes if there is a one-step transition of the machine which moves it between the configurations represented by these nodes. This example will come up again as a key tool in the proofs of Chapter 2.

The following fundamental theorem of automatic structures from [56], [11], [62] allows us to obtain new automatic structures from given ones. Consider the first-order logic extended by $\exists^\omega$ (there exist infinitely many) and $\exists^{n,m}$ (there exist n many mod m, where n and m are natural numbers) quantifiers. We denote this logic by FO^+.

Theorem 1.9 (Khoussainov, Nerode; 1995. Blumensath, Grädel; 1999. Khoussainov, Rubin, Stephan; 2004.)**.** *Suppose $\mathcal{A}$ is an automatic structure and $\varphi(\bar{x})$ is a formula in FO^+. There is an algorithm that produces an automaton that recognises exactly those tuples $\bar{a}$ from $\mathcal{A}$ that make φ true. In particular, the set of FO^+ sentences true in $\mathcal{A}$ is decidable.*

We present several constructions which use this theorem to demonstrate the closure of automata presentability under natural operations. Let $\mathcal{M}$ and $\mathcal{M}'$ be automata presentable structures over the same vocabulary with automatic presentations $\mathcal{A}, \mathcal{A}'$. The **disjoint union** of $\mathcal{M}$ and $\mathcal{M}'$ is defined to be the structure whose domain is the disjoint union of the domains of $\mathcal{M}, \mathcal{M}'$ and whose relations are disjoint unions of the relations in $\mathcal{A}, \mathcal{A}'$. An automatic presentation of the disjoint union is $\mathcal{B}$ where $B = A \times \{1\} \cup A' \times \{2\}$ (assuming that $1, 2$ are not in the alphabet of A, A'). We can extend this construction as follows [90]. The ω-fold disjoint union of a structure $\mathcal{A}$ is the disjoint union of ω many copies of $\mathcal{A}$.

Lemma 1.10 (Rubin; 2004)**.** *If $\mathcal{A}$ is automatic then its ω-fold disjoint union is automata presentable.*

Proof. Suppose that $\mathcal{A} = (A; R_1, R_2, \ldots)$ is automatic. Let $\mathcal{A}'$ be the structure with the same vocabulary whose domain is $A \times 1^*$ and where relations are defined by $\langle (x,i),(y,j)\rangle \in R'_m \iff i = j \;\&\; \langle x, y\rangle \in R_m$ for each m. It is clear that $\mathcal{A}'$ is automatic and is isomorphic to the ω-fold disjoint union of $\mathcal{A}$. $\square$

We now give some natural examples of automata presentable structures. Note that many of the automatic structures in the earlier examples arise as the presentations of the following automata presentable structures.

Example 1.11. Presburger arithmetic, the natural numbers under addition and order, have a finite automata presentation. The automata presentation here is the word structure $(\{0,1\}^* \cdot 1; +_2, \leq)$. This example will be discussed in detail in Chapter 4.

Example 1.12. Finitely generated abelian groups are all finite automata presentable. Recall that a group $\mathcal{G}$ is finitely generated if there is a finite subset S such that $\mathcal{G}$ is the smallest group containing S; a group $\mathcal{G}$ is called abelian if the group operation is commutative ($\forall a, b \in G \; a \cdot b = b \cdot a$). Since every such group is isomorphic to a finite direct sum of copies of $(\mathbb{Z}; +)$ and $(\mathbb{Z}_n; +)$ [89], and since each of these has an automata presentation, any finitely generated abelian group has an automata presentation. Note that free groups on more than one generator are not automata presentable [56], [11].

Example 1.13. The Boolean algebra of finite and co-finite subsets of ω is finite automata presentable. An automata presentation is the structure whose domain is $\{0,1\}^* \cup \{2,3\}^*$ where words in $\{0,1\}^*$ represent finite sets and words in $\{2,3\}^*$ represent cofinite sets. Note that by cardinality considerations it is immediate that the full Boolean algebra of all subsets of ω is not automata presentable.

Example 1.14. The linear order of the rational numbers $(\mathbb{Q}; \leq)$ has a finite automata presentation: $(\{0,1\}^* \cdot 1; \leq_{lex})$, where $u \leq_{lex} v$ is the lexicographic ordering and holds if and only if u is a prefix of v or $u = w0x$ and $v = w1y$ for some $w, x, y \in \{0,1\}^*$.

Example 1.15. An ordinal is a well-founded linear ordinal. All ordinals below ω^ω have finite automata presentations [56]. To see this, observe that ω has a finite

automata presentation and ω^n is first-order definable from ω. Likewise, sums of linear orders are first-order definable. An important result of Delhommé proves that this is a characterisation of ordinals with finite automata presentations [34]. This theorem is generalized in Chapter 2.

In the following, we abuse terminology and identify the notions of "automatic" and "automatically presentable" . The class of automatic structures is a proper subclass of the computable structures. We therefore mention some crucial definitions and facts about computable structures. Good references for the theory of computable structures include [43], [64].

Definition 1.16. A **computable structure** is $\mathcal{A} = (A; R_1, \ldots, R_m)$ whose domain and relations are all computable.

The domains of computable structures can always be identified with the set ω of natural numbers. Under this assumption, we introduce new constant symbols c_n for each $n \in \omega$ and interpret c_n as n. We expand the vocabulary of each structure to include these new constants c_n. In this context, $\mathcal{A}$ is computable if and only if the **atomic diagram** of $\mathcal{A}$ (the set of Gödel numbers of all quantifier-free sentences in the extended vocabulary that are true in $\mathcal{A}$) is a computable set. If $\mathcal{A}$ is computable and $\mathcal{B}$ is isomorphic to $\mathcal{A}$ then we say that $\mathcal{A}$ is a **computable presentation** of $\mathcal{B}$. Note that if $\mathcal{B}$ has a computable presentation then $\mathcal{B}$ has ω many computable presentations. In particular, a **computable ordinal** is a presentation of the natural numbers under a computable well-ordering. That is, it is the order-type of a computable well-ordering of the natural numbers. The least ordinal which is not computable is denoted ω_1^{CK} (after Church and Kleene).

1.3 Overview of complexity results

The following tables collect complexity results for the classes of computable structures, automatic structures, and some of their natural subclasses. Preceding each table are the relevant definitions. Complexity is measured with respect to the arithmetic hierarchy (see [88]). Recall that Δ_1^0 statements are computable and therefore such problems can be decided algorithmically. In the following tables, each problem is complete for its complexity class.

1.3.1 Isomorphism problem

Definition 1.17. The **isomorphism problem** for a class of structures $\mathcal{C}$ asks: given two elements of the class $\mathcal{C}$, are they isomorphic?

Table 1.1: Complexity of the isomorphism problem.

Isomorphism problem for computable structures [88]	Σ_1^1
Isomorphism problem for automatic structures [59]	Σ_1^1
Isomorphism problem for automatic graphs [53]	Σ_1^1
Isomorphism problem for locally finite automatic graphs [90]	Π_3^0
Isomorphism problem for automatic ordinals [63]	Δ_1^0
Isomorphism problem for automatic Boolean algebras [59]	Δ_1^0
Isomorphism problem for unary automatic linear orders [51]	Δ_1^0
Isomorphism problem for unary automatic trees [51]	Δ_1^0
Isomorphism problem for unary automatic equivalence relations [51]	Δ_1^0

1.3.2 Graph questions

In the following, we usually assume that graphs are infinite and undirected.

Definition 1.18. A graph is **locally finite** if each vertex has at most finitely many neighbours (vertices which share an edge). In other words, each vertex has finite valence.

Definition 1.19. A computable graph is called **highly recursive** if it is locally finite and the map $f : v \mapsto \{\text{neighbours of } v\}$ is computable.

We use the abbreviation **LFUA** to denote locally finite unary automatic graphs.

Definition 1.20. A graph is **Hamiltonian** if there is a path through the graph which visits every node exactly once.

Definition 1.21. A **clique** of a graph is a set of nodes, each pair of which is connected by an edge. That is, it is a subgraph which is a complete graph.

Definition 1.22. Given a graph and a vertex of the graph, the **infinity testing** problem asks whether the vertex lies in an infinite component of the graph.

Definition 1.23. The **infinite component** questions asks whether a given graph has some infinite component.

Definition 1.24. The **connectivity problem** asks if each pair vertices in a given graph is connected by a sequence of edges (a path). That is, it asks if the graph consists of a single connected component.

Definition 1.25. Given a graph and two vertices in the graph, the **reachability** problem asks whether there is a path in the graph which connects the two vertices.

Table 1.2: Complexity of graph theoretic questions

Is a highly recursive graph Hamiltonian? [48]	Σ_1^1
Is a planar automatic graph Hamiltonian? [69]	Σ_1^1
Does an automatic graph have an infinite clique? [91]	Δ_1^0
Infinite component problem for automatic graphs [90]	Σ_3^0
Infinity testing problem for automatic graphs [90]	Π_2^0
Connectivity problem for automatic graphs [90]	Π_2^0
Reachability problem for automatic graphs [90]	Σ_1^0
Infinite component problem for LFUA graphs [11],[52]	Δ_1^0
Infinity testing problem for LFUA graphs [11],[52]	Δ_1^0
Connectivity problem for LFUA graphs [11],[52]	Δ_1^0
Reachability problem for LFUA graphs [11],[52]	Δ_1^0

1.3.3 Tree questions

Definition 1.26. A **(partial order) tree** is a partially ordered set $(T; \leq)$ such that there is a $\leq$-least element of T, and each subset $\{x \in T : x \leq y\}$ is finite and is linearly ordered under $\leq$.

Definition 1.27. A **successor tree** is a structure $(T; S)$ such that the reflexive and transitive closure $\leq_S$ of S produces a partial order tree $(T; \leq_S)$.

1.4 Outline

We now outline the structure of the remainder of this thesis. Chapter 2 discusses various measures of complexity applied to automatic structures. In particular, set

Table 1.3: Complexity of tree questions

Does a recursive tree have an infinite path? [88]	Σ_1^1
Does an automatic successor tree have an infinite path? [69]	Σ_1^1
Does an automatic partial order tree have an infinite path? [63]	Δ_1^0

theoretic, topological, and model theoretic ranks are considered. In each setting, there is a natural bound on the ranks attained by computable structures. We show that automatic structures achieve ranks that are as high as those of computable structures. Thus, in this sense, automatic structures are as complicated as computable structures.

Chapter 3 restricts to a class of automatic structures which lend themselves more readily to efficient algorithms. In particular, we discuss locally finite unary automatic graphs. Several characterizations of this class of graphs are given. Then, polynomial-time algorithms are presented for the infinite component, infinity testing, connectivity, and reachability problems.

Finally, Chapter 4 discusses automata theoretic decision procedures. We recall linear programming questions and relate them to various logical theories. A known automata theoretic decision procedure for Presburger arithmetic is detailed, and then extended to suit the linear programming applications. The methodology is then applied for other mathematical structures including formal power series and the p-adics.

Chapter 2

Ranks

The results in this section were first reported in [53] and will appear in [55]. They reflect joint work with Bakhadyr Khoussainov.

2.1 Introduction

Most current results demonstrate that automatic structures are not complex in various concrete senses. However, in this chapter we use well-established concepts from both logic and model theory to prove results in the opposite direction. We now briefly describe the measures of complexity we use (ordinal heights of well-founded relations, Scott ranks of structures, and Cantor-Bendixson ranks of trees) and connect them with the results of this chapter.

A relation R is called **well-founded** if there is no infinite sequence $x_1, x_2, x_3, \ldots$ such that $(x_{i+1}, x_i) \in R$ for $i \in \omega$. In computer science, well-founded relations are of interest due to a natural connection between well-founded sets and terminating programs. We say that a program is **terminating** if every computation from an initial state is finite. This is equivalent to well-foundedness of the collection of states reachable from the initial state, under the reachability relation [10]. The **ordinal height** is a measure of the depth of well-founded relations. Since all automatic structures are also computable structures, the obvious bound for ordinal heights of automatic well-founded relations is ω_1^{CK} (the first non-computable ordinal). Sections 2.2 and 2.4 study the sharpness of this bound. Theorem 2.5 characterizes automatic well-founded partial orders in terms of their (relatively

low) ordinal heights, whereas Theorem 2.10 shows that ω_1^{CK} is the sharp bound in the general case. Note that Theorem 2.5 gives a generalization of Example 1.15 which dealt with well-founded linear orders rather than well-founded partial orders.

Theorem 2.5 *For each ordinal α, α is the ordinal height of an automatic well-founded partial order if and only if $\alpha < \omega^\omega$.*

Theorem 2.10 *For each (computable) ordinal $\alpha < \omega_1^{CK}$, there is an automatic well-founded relation $\mathcal{A}$ with ordinal height greater than α.*

In fact, given a computable ordinal α, Theorem 2.10 produces an automatic well-founded relation $\mathcal{A}$ whose ordinal height $r(\mathcal{A})$ satisfies $\alpha \leq r(\mathcal{A}) \leq \omega + \alpha$.

Section 2.5 is devoted to building automatic structures with high Scott ranks. The concept of Scott rank comes from a well-known theorem of Scott stating that every countable structure $\mathcal{A}$ may be associated to a sentence φ in $L_{\omega_1,\omega}$-logic which characterizes $\mathcal{A}$ up to isomorphism [92]. The minimal quantifier rank of such a formula is called the Scott rank of $\mathcal{A}$. Work by Nadel [77], Makkai [72], and Knight and Millar [67] gives information on possible Scott ranks for computable structures. In particular, we get that the upper bound on the Scott rank of computable structures is $\omega_1^{CK} + 1$. Since all automatic structures are computable structures, the Scott rank of any automatic structure can be at most $\omega_1^{CK} + 1$. But, until now, all the known examples of automatic structures had low Scott ranks. Results in [71], [34], [63] suggest that the Scott ranks of automatic structures could be bounded by small ordinals. This intuition is falsified in Section 2.5 with the theorem:

Theorem 2.20 *For each computable ordinal α there is an automatic structure of*

Scott rank at least α.

Moreover, for a given ordinal α less than or equal to ω_1^{CK}, Theorem 2.20 constructs an automatic structure $\mathcal{A}$ whose Scott rank is between α and $2+\alpha$. Thus, this theorem gives a new proof that the isomorphism problem for automatic structures is Σ_1^1-complete (another proof may be found in [59]).

In the last section of this chapter, we investigate the Cantor-Bendixson ranks of automatic trees. A **partial order tree** is a partially ordered set $(T;\leq)$ such that there is a $\leq$-least element of T, and each subset $\{x \in T : x \leq y\}$ is finite and is linearly ordered under $\leq$. A **successor tree** is a structure $(T;S)$ such that the reflexive and transitive closure $\leq_S$ of S produces a partial order tree $(T;\leq_S)$. The **derivative** of a tree $\mathcal{T}$ is obtained by removing all the nonbranching paths of the tree. One applies the derivative operation to $\mathcal{T}$ iteratively until a fixed point is reached. The least ordinal that is needed to reach the fixed point is called the **Cantor-Bendixson (CB) rank** of the tree. The CB rank plays an important role in logic, algebra, and topology. Informally, the CB rank tells us how far the structure is from algorithmically (or algebraically) simple structures. Again, the obvious bound on CB ranks of automatic successor trees is ω_1^{CK}. In [61], it is proved that the CB rank of any automatic partial order tree is finite and can be computed from the automaton for the $\leq$ relation on the tree. It has been an open question whether the CB ranks of automatic successor trees can also be bounded by small ordinals. We answer this question in the following theorem.

Theorem 2.28 *For $\alpha < \omega_1^{CK}$ there is an automatic successor tree of CB rank α.*

The main tool we use to prove results about high ranks is the configuration spaces of Turing machines, considered as automatic graphs. It is important to

note that graphs which arise as configuration spaces have very low model-theoretic complexity: their Scott ranks are at most 3, and if they are well-founded then their ordinal heights are at most ω (see Propositions 2.9 and 2.13). Hence, the configuration spaces serve merely as building blocks in the construction of automatic structures with high complexity, rather than contributing materially to the high complexity themselves.

2.2 Heights of automatic well-founded partial orders

In this section we consider structures $\mathcal{A} = (A; R)$ with a single binary relation. An element x is said to be **R-minimal for a set** X if for each $y \in X$, $(y, x) \notin R$. The relation R is said to be **well-founded** if every non-empty subset of A has an R-minimal element. This is equivalent to saying that $(A; R)$ has no infinite chains $x_1, x_2, x_3, \ldots$ where $(x_{i+1}, x_i) \in R$ for all i.

A **ranking function** for $\mathcal{A}$ is an ordinal-valued function f such that $f(y) < f(x)$ whenever $(y, x) \in R$. If f is a ranking function on $\mathcal{A}$, let $ord(f) = \sup\{f(x) : x \in A\}$. The structure $\mathcal{A}$ is well-founded if and only if $\mathcal{A}$ admits a ranking function. The **ordinal height** of $\mathcal{A}$, denoted $r(\mathcal{A})$, is the least ordinal α which is $ord(g)$ for some ranking function g on $\mathcal{A}$. An equivalent definition for the rank of $\mathcal{A}$ is the following. We define the function $r_{\mathcal{A}}$ by induction: for the R-minimal elements x, set $r_{\mathcal{A}}(x) = 0$; for z not R-minimal, put $r_{\mathcal{A}}(z) = \sup\{r(y) + 1 : (y, z) \in R\}$. Then $r_{\mathcal{A}}$ is a ranking function admitted by $\mathcal{A}$ and $r(\mathcal{A}) = \sup\{r_{\mathcal{A}}(x) : x \in A\}$. For $B \subseteq A$, we write $r(B)$ for the ordinal height of the structure obtained by restricting the relation R to the subset B.

Lemma 2.1. *If $\alpha < \omega_1^{CK}$, there is a computable well-founded relation of ordinal*

height α.

Proof. This lemma is trivial: the ordinal height of an ordinal α is α itself. Since all computable ordinals are computable and well-founded relations, we are done. □

The next lemma follows easily from the well-foundedness of ordinals and of R.

Lemma 2.2. *For a structure* $\mathcal{A} = (A; R)$ *where* R *is well-founded, if* $r(\mathcal{A}) = \alpha$ *and* $\beta < \alpha$ *then there is an* $x \in A$ *such that* $r_{\mathcal{A}}(x) = \beta$.

Proof. We proceed by induction. If $\alpha = 0$ the claim is vacuous; if $\alpha = 1$ the only possible value for β is 0, which must be attained by definition of height for R-minimal elements of A. For the inductive step, let $\alpha > 1$ and $\beta < \alpha$. Consider the sets $L_\beta = \{x \in A : r_{\mathcal{A}}(x) < \beta\}$ and $R_\beta = \{x \in A : r_{\mathcal{A}}(x) > \beta\}$. Suppose, on the one hand, that $R_\beta = \emptyset$. Then $\sup\{r_{\mathcal{A}}(x) : x \in A\} \leq \sup\{\beta, r_{\mathcal{A}}(x) : x \in L_\beta\} \leq \beta$, a contradiction with $r(\mathcal{A}) = \alpha > \beta$. Therefore, $R_\beta \neq \emptyset$ and let $y \in R_\beta$ be an element such that $r_{\mathcal{A}}(y) \leq r_{\mathcal{A}}(w)$ for all $w \in R_\beta$ (by well-foundedness of the ordinals). Hence, for all $w \in R_\beta$, it is not the case that $R(w, y)$. Assume for a contradiction that there is no $z \in A$ with $r_{\mathcal{A}}(z) = \beta$. Then,

$$r_{\mathcal{A}}(y) = \sup\{r_A(x) + 1 : R(x, y)\} = \sup\{r_A(x) + 1 : x \in L_\beta \ \&\ R(x, y)\} < \beta + 1.$$

In particular, we have that $r_{\mathcal{A}}(y) \leq \beta$, a contradiction with $y \in R_\beta$. □

For the remainder of this section, we assume further that R is a partial order. For convenience, we write $\leq$ instead of R. Thus, we consider automatic well-founded partial orders $\mathcal{A} = (A; \leq)$. We will use the notion of **natural sum of ordinals**. The natural sum of ordinals α, β (denoted $\alpha +' \beta$) is defined recursively: $\alpha +' 0 = \alpha$, $0 +' \beta = \beta$, and $\alpha +' \beta$ is the least ordinal strictly greater than $\gamma +' \beta$ for all $\gamma < \alpha$ and strictly greater than $\alpha +' \gamma$ for all $\gamma < \beta$.

Lemma 2.3. *Let A_1 and A_2 be disjoint subsets of A such that $A = A_1 \cup A_2$. Consider the partially ordered sets $\mathcal{A}_1 = (A_1; \leq_1)$ and $\mathcal{A}_2 = (A_2; \leq_2)$ obtained by restricting $\leq$ to A_1 and A_2 respectively. Then, $r(\mathcal{A}) \leq \alpha_1 +' \alpha_2$, where $\alpha_i = r(\mathcal{A}_i)$.*

Proof. We will show that there is a ranking function on A whose range is contained in the ordinal $\alpha_1 +' \alpha_2$. For each $x \in A$ consider the partially ordered sets $\mathcal{A}_{1,x}$ and $\mathcal{A}_{2,x}$ obtained by restricting $\leq$ to $\{z \in A_1 : z < x\}$ and $\{z \in A_2 : z < x\}$, respectively. Define $f(x) = r(\mathcal{A}_{1,x}) +' r(\mathcal{A}_{2,x})$. We claim that f is a ranking function. Indeed, assume that $x < y$. Then, since $\leq$ is transitive, it must be the case that $\mathcal{A}_{1,x} \subseteq A_{1,y}$ and $\mathcal{A}_{2,x} \subseteq A_{2,y}$. Therefore, $r(\mathcal{A}_{1,x}) \leq r(\mathcal{A}_{1,y})$ and $r(\mathcal{A}_{2,x}) \leq r(\mathcal{A}_{2,y})$. At least one of these inequalities must be strict. To see this, assume that $x \in A_1$ (the case $x \in A_2$ is similar). Then since $x \in A_{1,y}$, it is the case that $r(\mathcal{A}_{1,x}) + 1 \leq r(\mathcal{A}_{1,y})$ by the definition of ranks. Therefore, we have that $f(x) < f(y)$. Moreover, the image of $f(x)$ is contained in $\alpha_1 +' \alpha_2$. □

Corollary 2.4. *If $r(\mathcal{A}) = \omega^n$ and $A = A_1 \cup A_2$ where $A_1 \cap A_2 = \emptyset$, then either $r(\mathcal{A}_1) = \omega^n$ or $r(\mathcal{A}_2) = \omega^n$.*

Khoussainov and Nerode [56] show that, for each n, there is a finite automata presentation of the ordinal ω^n. It is clear that such a presentation has ordinal height ω^n. The next theorem proves that ω^ω is the sharp bound on ranks of all automatic well-founded partial orders. Once Corollary 2.4 has been established, the proof of Theorem 2.5 follows Delhommé [34] and Rubin [90].

Theorem 2.5. *For each ordinal α, α is the ordinal height of an automatic well-founded partial order if and only if $\alpha < \omega^\omega$.*

Proof. One direction of the proof is immediate from [56] (see Example 1.15). For

the other direction, assume for a contradiction that there is an automatic well-founded partial order $\mathcal{A} = (A; \leq)$ with $r(\mathcal{A}) = \alpha \geq \omega^\omega$. Let $(S_A, \iota_A, \Delta_A, F_A)$ and $(S_\leq, \iota_\leq, \Delta_\leq, F_\leq)$ be finite automata over Σ recognising A and $\leq$ (respectively). By Lemma 2.2, for each $n > 0$ there is $u_n \in A$ such that $r_\mathcal{A}(u_n) = \omega^n$. For each $u \in A$ we define the set

$$u \downarrow = \{x \in A : x < u\}.$$

Note that if $r_\mathcal{A}(u)$ is a limit ordinal then $r_\mathcal{A}(u) = r(u \downarrow)$. We define a finite partition of $u \downarrow$ in order to apply Corollary 2.4. To do so, for $u, v \in \Sigma^*$, define $X_v^u = \{vw \in A : w \in \Sigma^* \ \& \ vw < u\}$. Each set of the form $u \downarrow$ can then be partitioned based on the prefixes of words as follows:

$$u \downarrow = \{x \in A : |x| < |u| \ \& \ x < u\} \cup \bigcup_{v \in \Sigma^* : |v| = |u|} X_v^u.$$

(All the unions above are finite and disjoint.) Hence, applying Corollary 2.4, for each u_n there exists a v_n such that $|u_n| = |v_n|$ and $r(X_{v_n}^{u_n}) = r(u_n \downarrow) = \omega^n$.

Meanwhile, we use the automata presenting $\mathcal{A}$ to define the following equivalence relation on pairs of words of equal lengths:

$$\begin{aligned}(u, v) \sim (u', v') \iff & \Delta_A(\iota_A, v) = \Delta_A(\iota_A, v') \ \& \\ & \Delta_\leq(\iota_\leq, \binom{v}{u}) = \Delta_\leq(\iota_\leq, \binom{v'}{u'})\end{aligned}$$

There are at most $|S_A| \times |S_\leq|$ equivalence classes. Thus, the infinite sequence (u_1, v_1), (u_2, v_2), ... contains m, n such that $m \neq n$ and $(u_m, v_m) \sim (u_n, v_n)$.

Lemma 2.6. *For any $u, v, u', v' \in \Sigma^*$, if $(u, v) \sim (u', v')$ then $r(X_v^u) = r(X_{v'}^{u'})$.*

To prove the lemma, consider $g : X_v^u \to X_{v'}^{u'}$ defined as $g(vw) = v'w$. From the equivalence relation, we see that g is well-defined, bijective, and order preserving. Hence $X_v^u \cong X_{v'}^{u'}$ (as partial orders). Therefore, $r(X_v^u) = r(X_{v'}^{u'})$.

By Lemma 2.6, $\omega^m = r(X^{u_m}_{v_m}) = r(X^{u_n}_{v_n}) = \omega^n$, a contradiction with the assumption that $m \neq n$. Therefore, there is no automatic well-founded partial order of ordinal height greater than or equal to ω^ω. □

2.3 Configuration spaces of Turing machines

In the forthcoming constructions, we embed computable structures into automatic ones via configuration spaces of Turing machines. This subsection provides terminology and background for these constructions. Let $\mathcal{M}$ be an n-tape deterministic Turing machine. The **configuration space** of $\mathcal{M}$, denoted by $Conf(\mathcal{M})$, is a directed graph whose nodes are configurations of $\mathcal{M}$. The nodes are n-tuples, each of whose coordinates represents the contents of a tape. Each tape is encoded as $(w\ q\ w')$, where $w, w' \in \Sigma^*$ are the symbols on the tape before and after the location of the read/write head, and q is one of the states of $\mathcal{M}$. The edges of the graph are all the pairs of the form (c_1, c_2) such that there is an instruction of $\mathcal{M}$ that transforms c_1 to c_2 in one step. The configuration space is an automatic graph. The out-degree of every vertex in $Conf(\mathcal{M})$ is 1; the in-degree need not be 1. Later, it will be helpful to have some control over the structure of the configuration space of a given Turing machine. The following definition and theorem from [8] give us this control.

Definition 2.7. A deterministic Turing machine $\mathcal{M}$ is **reversible** if $Conf(\mathcal{M})$ consists only of finite chains and chains of type ω.

Lemma 2.8 (Bennett; 1973)**.** *For any deterministic* 1*-tape Turing machine there is a reversible* 3*-tape Turing machine which accepts the same language.*

Proof. (Sketch) Given a deterministic Turing machine, define a 3-tape Turing ma-

chine with a modified set of instructions. The modified instructions have the property that neither the domains nor the ranges overlap. The first tape performs the computation exactly as the original machine would have done. As the new machine executes each instruction, it stores the index of the instruction on the second tape, forming a history. Once the machine enters a state which would have been halting for the original machine, the output of the computation is copied onto the third tape. Then, the machine runs the computation backwards and erases the history tape. The halting configuration contains the input on the first tape, blanks on the second tape, and the output on the third tape. □

We establish the following notation for a 3-tape reversible Turing machine $\mathcal{M}$ given by the construction in this lemma. A **valid initial configuration** of $\mathcal{M}$ is of the form $(\lambda\ \iota\ x, \lambda, \lambda)$, where x is in the domain, λ is the empty string, and ι is the initial state of $\mathcal{M}$. From the proof of Lemma 2.8, observe that a **final (halting) configuration** is of the form $(x, \lambda, \lambda\ q_f\ y)$, with q_f a halting state of $\mathcal{M}$. Also, because of the reversibility assumption, all the chains in $Conf(\mathcal{M})$ are either finite or ω-chains (the order type of the natural numbers). In particular, this means that $Conf(\mathcal{M})$ is well-founded under the edge relation. We call an element of in-degree 0 a **base** (of a chain). The set of valid initial or final configurations is regular. We classify the components (chains) of $Conf(\mathcal{M})$ as follows:

- **Terminating computation chains**: finite chains whose base is a valid initial configuration; that is, one of the form $(\lambda\ \iota\ x, \lambda, \lambda)$, for $x \in \Sigma^*$.
- **Non-terminating computation chains**: infinite chains whose base is a valid initial configuration.
- **Unproductive chains**: chains whose base is not a valid initial configuration.

Configuration spaces of reversible Turing machines are locally finite graphs (graphs of finite degree) which are well-founded. Hence, the following proposition guarantees that their ordinal heights are small.

Proposition 2.9. *If $G = (A; E)$ is a locally finite graph then either E is well-founded and the ordinal height of E is not above ω, or E is not well-founded.*

Proof. Suppose G is a locally finite graph and E is well-founded. For a contradiction, suppose $r(G) > \omega$. Then there is $v \in A$ with $r(v) = \omega$. By definition, $r(v) = \sup\{r(u) : uEv\}$. But, this implies that there are infinitely many elements E-below v, a contradiction with local finiteness of G. □

2.4 Heights of automatic well-founded relations

We are now ready to prove that ω_1^{CK} is the sharp bound for ordinal heights of automatic well-founded relations.

Theorem 2.10. *For each computable ordinal $\alpha < \omega_1^{CK}$, there is an automatic well-founded relation $\mathcal{A}$ with ordinal height greater than α. In particular, $\alpha \leq r(\mathcal{A}) \leq \omega + \alpha$.*

Proof. The proof of this theorem uses properties of Turing machines and their configuration spaces. We take a computable well-founded relation whose ordinal height is α, and "embed" it into an automatic well-founded relation with similar ordinal height.

By Lemma 2.1, let $\mathcal{C} = (C; L_\alpha)$ be a computable well-founded relation of ordinal height α. We assume without loss of generality that $C = \Sigma^*$ for some finite

alphabet Σ. Let $\mathcal{M}$ be the Turing machine computing the relation L_α. On each pair (x, y) from the domain, $\mathcal{M}$ halts and outputs "yes" or "no" . By Lemma 2.8, we can assume that $\mathcal{M}$ is reversible. Recall that $Conf(\mathcal{M}) = (D; E)$ is an automatic graph. We define the domain of our automatic structure to be $A = \Sigma^* \cup D$. The binary relation of the automatic structure is:

$$R = E \ \cup \ \{(x, (\lambda \ \iota \ (x, y), \lambda, \lambda)) : x, y \in \Sigma^*\} \cup$$
$$\{(((x, y), \lambda, \lambda \ q_f \ \text{"yes"} \ ), y) : x, y \in \Sigma^*\}.$$

Intuitively, the structure $(A; R)$ is a stretched out version of $(C; L_\alpha)$ with infinitely many finite pieces extending from elements of C, and with disjoint pieces which are either finite chains or chains of type ω. The structure $(A; R)$ is automatic because its domain is a regular set of words and the relation R is recognisable by a 2-tape automaton. We should verify, however, that R is well-founded. Let $Y \subset A$. If $Y \cap C \neq \emptyset$ then since $(C; L_\alpha)$ is well-founded, there is $x \in Y \cap C$ which is L_α-minimal. The only possible elements u in Y for which $(u, x) \in R$ are those which lie on computation chains connecting some $z \in C$ with x. Since each such computation chain is finite, there is an R-minimal u below x on each chain. Any such u is R-minimal for Y. On the other hand, if $Y \cap C = \emptyset$, then Y consists of disjoint finite chains and chains of type ω. Any such chain has a minimal element, and any of these elements are R-minimal for Y. Therefore, $(A; R)$ is an automatic well-founded structure.

We now consider the ordinal height of $(A; R)$. For each element $x \in C$, an easy induction on $r_C(x)$, shows that $r_{\mathcal{C}}(x) \leq r_{\mathcal{A}}(x) \leq \omega + r_{\mathcal{C}}(x)$. We denote by $\ell(a, b)$ the (finite) length of the computation chain of $\mathcal{M}$ with input (a, b). For any element $a_{x,y}$ in the computation chain which represents the computation of $\mathcal{M}$ determining whether $(x, y) \in R$, we have $r_{\mathcal{A}}(x) \leq r_{\mathcal{A}}(a_{x,y}) \leq r_{\mathcal{A}}(x) + \ell(x, y)$. For

any element u in an unproductive chain of the configuration space, $0 \leq r_{\mathcal{A}}(u) < \omega$. Therefore, since $C \subset A$, $r(\mathcal{C}) \leq r(\mathcal{A}) \leq \omega + r(C)$. □

2.5 Automatic structures and Scott rank

The Scott rank of a structure is introduced in the proof of Scott's Isomorphism Theorem [92]. Since then, variants of the Scott rank have been used in the computable model theory literature. We follow the definition of Scott rank from [24].

Definition 2.11. For structure $\mathcal{A}$ and tuples $\bar{a}, \bar{b} \in A^n$ (of equal length), define

- $\bar{a} \equiv^0 \bar{b}$ if $\bar{a}, \bar{b}$ satisfy the same quantifier-free formulas in the language of $\mathcal{A}$;
- For $\alpha > 0$, $\bar{a} \equiv^\alpha \bar{b}$ if for all $\beta < \alpha$, for each $\bar{c}$ (of arbitrary length) there is $\bar{d}$ such that $\bar{a}, \bar{c} \equiv^\beta \bar{b}, \bar{d}$; and for each $\bar{d}$ (of arbitrary length) there is $\bar{c}$ such that $\bar{a}, \bar{c} \equiv^\beta \bar{b}, \bar{d}$.

Then, the **Scott rank** of the tuple $\bar{a}$, denoted by $\mathcal{SR}(\bar{a})$, is the least β such that for all $\bar{b} \in A^n$, $\bar{a} \equiv^\beta \bar{b}$ implies that $(\mathcal{A}, \bar{a}) \cong (\mathcal{A}, \bar{b})$. The Scott rank of $\mathcal{A}$, denoted by $\mathcal{SR}(\mathcal{A})$, is the least α greater than the Scott ranks of all tuples of $\mathcal{A}$.

Example 2.12. $\mathcal{SR}(\mathbb{Q}; \leq) = 1$, $\mathcal{SR}(\omega; \leq) = 2$, and $\mathcal{SR}(n \cdot \omega; \leq) = n + 1$.

Configuration spaces of reversible Turing machines are locally finite graphs. By the proposition below, they all have low Scott Rank.

Proposition 2.13. *If $G = (V; E)$ is a locally finite graph, $SR(G) \leq 3$.*

Proof. The n-neighbourhood of a subset U, denoted $B_n(U)$, is defined as follows: $B_0(U) = U$ and $B_n(U)$ is the set of $v \in V$ which can be reached from U by n or fewer edges. The proof of Proposition 2.13 relies on two lemmas.

Lemma 2.14. *Let $\bar{a}, \bar{b} \in V$ be such that $\bar{a} \equiv^2 \bar{b}$. Then, for all n, there is a bijection of the n-neighbourhoods around $\bar{a}, \bar{b}$ which sends $\bar{a}$ to $\bar{b}$ and which respects E.*

Proof. For a given n, let $\bar{c} = B_n(\bar{a}) \setminus \bar{a}$. Note that $\bar{c}$ is a finite tuple because of the local finiteness condition. Since $\bar{a} \equiv^2 \bar{b}$, there is $\bar{d}$ such that $\bar{a}\bar{c} \equiv^1 \bar{b}\bar{d}$. It suffices to show that $B_n(\bar{b}) = \bar{b}\bar{d}$. Hence, two set inclusions are needed. First, we show that $d_i \in B_n(\bar{b})$. By definition, we have that $c_i \in B_n(\bar{a})$, and let $a_j, u_1, \ldots, u_{n-1}$ witness this. Then since $\bar{a}\bar{c} \equiv^1 \bar{b}\bar{d}$, there are $v_1, \ldots, v_{n-1}$ such that $\bar{a}\bar{c}\bar{u} \equiv^0 \bar{b}\bar{d}\bar{v}$. In particular, we have that if $c_i E u_i E \cdots E u_{n-1} E a_j$, then also $d_i E v_i E \cdots E v_{n-1} E b_j$ (and likewise if the E relation is in the other direction). Hence, $d_i \in B_n(\bar{b})$. Conversely, suppose $v \in B_n(\bar{b}) \setminus \bar{d}$. Let $v_1, \ldots, v_n$ be witnesses and this will let us find a new element of $B_n(\bar{a})$ which is not in $\bar{c}$, a contradiction. □

Lemma 2.15. *Let $G = (V; E)$ be a graph. Suppose $\bar{a}, \bar{b} \in V$ are such that for all n, $(B_n(\bar{a}); E, \bar{a}) \cong (B_n(\bar{b}); E, \bar{b})$. Then there is an isomorphism between the component of G containing $\bar{a}$ and that containing $\bar{b}$ which sends $\bar{a}$ to $\bar{b}$.*

Proof. We consider a tree of partial isomorphisms of G. The nodes of the tree are bijections from $B_n(\bar{a})$ to $B_n(\bar{b})$ which respect the relation E and map $\bar{a}$ to $\bar{b}$. Node f is the child of node g in the tree if $\mathrm{dom}(f) = B_n(\bar{a})$, $\mathrm{dom}(g) = B_{n+1}(\bar{a})$ and $f \supset g$. Note that the root of this tree is the map which sends $\bar{a}$ to $\bar{b}$. Moreover, the tree is finitely branching and is infinite by Lemma 2.14. Therefore, König's Lemma gives an infinite path through this tree. The union of all partial isomorphisms along this path is the required isomorphism. □

To finish the proof of Proposition 2.13, we note that for any $\bar{a}, \bar{b}$ in V such that $\bar{a} \equiv^2 \bar{b}$, Lemmas 2.14 and 2.15 yield an isomorphism from the component of $\bar{a}$ to the component of $\bar{b}$ that maps $\bar{a}$ to $\bar{b}$. Hence, if $\bar{a} \equiv^2 \bar{b}$, there is an automorphism of G that maps $\bar{a}$ to $\bar{b}$. Therefore, for each $\bar{a} \in V$, $SR(\bar{a}) \leq 2$, so $SR(G) \leq 3$. □

Let $\mathcal{C} = (C; R_1, \ldots, R_m)$ be a computable structure. Recall that since C is a computable set, we may assume it is Σ^* for some finite alphabet Σ. We construct an automatic structure $\mathcal{A}$ whose Scott rank is (close to) the Scott rank of $\mathcal{C}$. The construction of $\mathcal{A}$ involves connecting the configuration spaces of Turing machines computing relations $R_1, \ldots, R_m$ with each other and with Σ^*. Note that Proposition 2.13 suggests that ensuring that the Scott rank of the resulting automatic structure is sufficiently high constitutes the main part of the construction because each of the configuration spaces has low Scott rank. The construction in some sense expands $\mathcal{C}$ into an automatic structure. We comment that expansions do not necessarily preserve the Scott rank. For example, any computable structure has an expansion with Scott rank 2 obtained by adding the successor relation into the signature.

We detail the construction for R_i. Let $\mathcal{M}_i$ be a Turing machine for R_i. By a simple modification of the machine we assume that $\mathcal{M}_i$ halts if and only if its output is "yes" . By Lemma 2.8, we can also assume that $\mathcal{M}_i$ is reversible. We now modify the configuration space $Conf(\mathcal{M}_i)$ so as to respect the isomorphism type of $\mathcal{C}$. This will ensure that the construction (almost) preserves the Scott rank of $\mathcal{C}$. We use the terminology from Subsection 2.3.

Smoothing out unproductive parts. The length and number of unproductive chains is determined by the machine $\mathcal{M}_i$ and hence may differ even for Turing machines computing the same set. In this stage, we standardize the format of

this unproductive part of the configuration space. We wish to add enough redundant information in the unproductive section of the structure so that if two given computable structures are isomorphic, the unproductive parts of the automatic representations will also be isomorphic . We add countably infinitely many chains of length n (for each n) and countably infinitely many copies of ω. This ensures that the (smoothed) unproductive section of the configuration space of any Turing machine will be isomorphic and preserves automaticity. We comment that adding this redundancy preserves automaticity since the operation is a disjoint union of automatic structures.

Smoothing out lengths of computation chains. We turn our attention to the chains which have valid initial configurations at their base. The length of each finite chain denotes the length of computation required to return a "yes" answer. We will smooth out these chains by adding "fans" to each base. For this, we connect to each base of a computation chain a structure which consists of countably infinitely many chains of each finite length. To do so we follow Rubin [90]: consider the structure whose domain is 0^*01^* and whose relation is given by xEy if and only if $|x| = |y|$ and y is the least lexicographic successor of x. This structure has a finite chain of every finite length. As in Lemma 1.10, we take the ω-fold disjoint union of the structure and identify the bases of all the finite chains. We get a "fan" with infinitely many chains of each finite size whose base can be identified with a valid initial computation state. Also, the fan has an infinite component if and only if R_i does not hold of the input tuple corresponding to the base. The result is an automatic graph, $Smooth(R_i) = (D_i; E_i)$, which extends $Conf(\mathcal{M}_i)$.

Connecting domain symbols to the computations of the relation. We apply the construction above to each R_i in the signature of $\mathcal{C}$. Taking the union

of the resulting automatic graphs and adding vertices for the domain, we have the structure $(\Sigma^* \cup \cup_i D_i; E_1, \ldots, E_n)$ (where we assume that the D_i are disjoint). We assume without loss of generality that each $\mathcal{M}_i$ has a different initial state, denoted by ι_i. We add n predicates F_i to the signature of the automatic structure. These predicates connect the elements of the domain of $\mathcal{C}$ with the computations of the relations R_i:

$$F_i = \{(x_0, \ldots, x_{m_i-1}, (\lambda\ \iota_i\ (x_0, \ldots, x_{m_i-1}), \lambda, \lambda)) : x_0, \ldots, x_{m_i-1} \in \Sigma^*\}.$$

Note that for $\bar{x} \in \Sigma^*$, $R_i(\bar{x})$ is true if and only if $F_i(\bar{x}, (\lambda\ \iota_i\ \bar{x}, \lambda, \lambda))$ holds and all E_i chains emanating from $(\lambda\ \iota_i\ \bar{x}, \lambda, \lambda)$ are finite. We have now concluded building the automatic structure

$$\mathcal{A} = (\Sigma^* \cup \cup_i D_i; E_1, \ldots, E_n, F_1, \ldots, F_n).$$

Two technical lemmas are used to show that the Scott rank of $\mathcal{A}$ is close to α:

Lemma 2.16. *For $\bar{x}, \bar{y} \in \Sigma^*$ and for ordinal α, if $\bar{x} \equiv^\alpha_{\mathcal{C}} \bar{y}$ then $\bar{x} \equiv^\alpha_{\mathcal{A}} \bar{y}$.*

Proof. Let $X = \mathrm{dom}\mathcal{A} \setminus \Sigma^*$. We prove the stronger result that for any ordinal α, and for all $\bar{x}, \bar{y} \in \Sigma^*$ and $\bar{x}', \bar{y}' \in X$, if the following assumptions hold

1. $\bar{x} \equiv^\alpha_{\mathcal{C}} \bar{y}$;
2. $\langle \bar{x}', E_i : i = 1 \ldots n \rangle_{\mathcal{A}} \cong_f \langle \bar{y}', E_i, : i = 1 \ldots n \rangle_{\mathcal{A}}$ (hence the substructures generated by $\bar{x}'$ and $\bar{y}'$ in A are isomorphic) with $f(\bar{x}') = \bar{y}'$; and
3. for each $x'_k \in \bar{x}'$, $i = 1, \ldots, n$ and subsequence of indices of length m_i,

$$x'_k = (\lambda\ \iota_i\ \bar{x}_j, \lambda, \lambda) \quad \Longleftrightarrow \quad y'_k = (\lambda\ \iota_i\ \bar{y}_j, \lambda, \lambda)$$

then $\bar{x}\bar{x}' \equiv^\alpha_{\mathcal{A}} \bar{y}\bar{y}'$. The lemma follows if we take $\bar{x}' = \bar{y}' = \lambda$ (the empty string).

The proof of this stronger statement goes by induction on α. If $\alpha = 0$, we need to show that for each $i, k, k', k_0, \ldots, k_{m_i-1}$,

$$(*) \qquad E_i(x'_k, x'_{k'}) \iff E_i(y'_k, y'_{k'}),$$

and that

$$(**) \qquad F_i(x_{k_0}, \ldots, x_{k_{m_i-1}}, x'_{k'}) \iff F_i(y_{k_0}, \ldots, y_{k_{m_i-1}}, y'_{k'}).$$

Statement $(*)$ follows by assumption 2, since the isomorphism must preserve the E_i relations and maps $\bar{x}'$ to $\bar{y}'$. Statement $(**)$ follows by assumption 3.

Assume now that $\alpha > 0$ and that the result holds for all $\beta < \alpha$. Let $\bar{x}, \bar{y} \in \Sigma^*$ and $\bar{x}', \bar{y}' \in A$ satisfy assumptions 1, 2, and 3. We will show that $\bar{x}\bar{x}' \equiv^{\alpha}_{\mathcal{A}} \bar{y}\bar{y}'$. Let $\beta < \alpha$ and suppose $\bar{u} \in \Sigma^*, \bar{u}' \in A$. By assumption 1, there is $\bar{v} \in \Sigma^*$ such that $\bar{x}\bar{u} \equiv^{\beta}_{\mathcal{C}} \bar{y}\bar{v}$. By the construction (in particular, the smoothing steps), we can find a corresponding $\bar{v}' \in A$ such that assumptions 2, 3 hold. Applying the inductive hypothesis, we get that $\bar{x}\bar{u}\bar{x}'\bar{u}' \equiv^{\beta}_{\mathcal{A}} \bar{y}\bar{v}\bar{y}'\bar{v}'$. Analogously, given $\bar{v}, \bar{v}'$ we can find the necessary $\bar{u}, \bar{u}'$. Therefore, $\bar{x}\bar{x}' \equiv^{\alpha}_{\mathcal{A}} \bar{y}\bar{y}'$. □

Lemma 2.17. *If $\bar{x} \in \Sigma^* \cup \cup_i D_i$, there is $\bar{y} \in \Sigma^*$ with $\mathcal{SR}_{\mathcal{A}}(\bar{x}\bar{x}'\bar{u}) \leq 2 + \mathcal{SR}_{\mathcal{C}}(\bar{y})$.*

Proof. Let X_P denote the subset of $X = A \setminus \Sigma^*$ which corresponds to elements on fans associated with productive chains of the configuration space. Let X_U denote the subset of X containing elements of the unproductive chains of the configuration space. Thus, $A = \Sigma^* \cup X_P \cup X_U$, a disjoint union. We will show that for each $\bar{x} \in \Sigma^*$, $\bar{x}' \in X_P$, $\bar{u} \in X_U$ there is $\bar{y} \in \Sigma^*$ such that $\mathcal{SR}_{\mathcal{A}}(\bar{x}\bar{x}'\bar{u}) \leq 2 + \mathcal{SR}_{\mathcal{C}}(\bar{y})$.

Given $\bar{x}, \bar{x}', \bar{u}$, let $\bar{y} \in \Sigma^*$ be a minimal element satisfying that $\bar{x} \subset \bar{y}$ and that $\bar{x}' \subset \langle \bar{y}, E_i, F_i : i = 1 \ldots n \rangle_{\mathcal{A}}$. We will show that $\bar{y}$ is the desired witness. First, we observe that since the unproductive part of the structure is disconnected from the

productive elements we can consider the two independently. Moreover, because the structure of the unproductive part is predetermined and simple, for $\bar{u}, \bar{v} \in X_U$, if $\bar{u} \equiv^1_{\mathcal{A}} \bar{v}$ then $(\mathcal{A}, \bar{u}) \cong (\mathcal{A}, \bar{v})$. It remains to consider the productive part of the structure.

Consider any $\bar{z} \in \Sigma^*$, $\bar{z}' \in X_P$ satisfying $\bar{z}' \subset \langle \bar{z}, E_i, F_i : i = 1 \ldots n \rangle_{\mathcal{A}}$. We claim that $SR_{\mathcal{A}}(\bar{z}\bar{z}') \leq 2 + \mathcal{SR}_{\mathcal{C}}(\bar{z})$. It suffices to show that for all α, for all $\bar{w} \in \Sigma^*, \bar{w}' \in X_P$,

$$\bar{z}\bar{z}' \equiv^{2+\alpha}_{\mathcal{A}} \bar{w}\bar{w}' \qquad \Longrightarrow \qquad \bar{z} \equiv^{\alpha}_{\mathcal{C}} \bar{w}.$$

This is sufficient for the following reason. If $\bar{z}\bar{z}' \equiv^{2+\mathcal{SR}_{\mathcal{C}}(\bar{z})}_{\mathcal{A}} \bar{w}\bar{w}'$ then $\bar{z} \equiv^{\mathcal{SR}_{\mathcal{C}}(\bar{z})}_{\mathcal{C}} \bar{w}$ and hence $(\mathcal{C}, \bar{z}) \cong (\mathcal{C}, \bar{w})$. From this automorphism, we can define an automorphism of $\mathcal{A}$ mapping $\bar{z}\bar{z}'$ to $\bar{w}\bar{w}'$ because $\bar{z}\bar{z}' \equiv^2_{\mathcal{A}} \bar{w}\bar{w}'$ and hence for each i, the relative positions of $\bar{z}'$ and $\bar{w}'$ in the fans above $\bar{z}$ and $\bar{w}$ are isomorphic. Therefore, $2 + \mathcal{SR}_{\mathcal{C}}(\bar{z}) \geq \mathcal{SR}_{\mathcal{A}}(\bar{z}\bar{z}')$.

So, we now show that for all α, for all $\bar{w} \in \Sigma^*, \bar{w}' \in X_P$, $\bar{z}\bar{z}' \equiv^{2+\alpha}_{\mathcal{A}} \bar{w}\bar{w}'$ implies that $\bar{z} \equiv^{\alpha}_{\mathcal{C}} \bar{w}$. We proceed by induction on α. For $\alpha = 0$, suppose that $\bar{z}\bar{z}' \equiv^2_{\mathcal{A}} \bar{w}\bar{w}'$. This implies that for each i and for each subsequence of length m_i of the indices, the E_i-fan above $\bar{z}_j$ has an infinite chain if and only if the E_i-fan above $\bar{w}_j$ does. Therefore, $R_i(\bar{z}_j)$ if and only if $R_i(\bar{w}_j)$. Hence, $\bar{z} \equiv^0_{\mathcal{C}} \bar{w}$, as required. For the inductive step, we assume the result holds for all $\beta < \alpha$. Suppose that $\bar{z}\bar{z}' \equiv^{2+\alpha}_{\mathcal{A}} \bar{w}\bar{w}'$. Let $\beta < \alpha$ and $\bar{c} \in \Sigma^*$. Then $2 + \beta < 2 + \alpha$ so by definition there is $\bar{d} \in \Sigma^*, \bar{d}' \in X_P$ such that $\bar{z}\bar{z}'\bar{c} \equiv^{2+\beta}_{\mathcal{A}} \bar{w}\bar{w}'\bar{d}\bar{d}'$. However, since $2 + \beta > 1$, $\bar{d}'$ must be empty (elements in Σ^* cannot be 1-equivalent to elements in X_P). By the induction hypothesis, $\bar{z}\bar{c} \equiv^{\beta}_{\mathcal{C}} \bar{w}\bar{d}$. The argument works symmetrically if we are given $\bar{d}$ and want to find $\bar{c}$. Thus, $\bar{z} \equiv^{\alpha}_{\mathcal{C}} \bar{w}$, as required. □

Lemmas 2.16 and 2.17 may be combined to prove the main result about our construction.

Theorem 2.18. *Let $\mathcal{C}$ be a computable structure and construct the automatic structure $\mathcal{A}$ from it as above. Then $\mathcal{SR}(\mathcal{C}) \leq \mathcal{SR}(\mathcal{A}) \leq 2 + \mathcal{SR}(\mathcal{C})$.*

Proof. Let $\bar{x}$ be a tuple in the domain of $\mathcal{C}$. By the definition of Scott rank, $\mathcal{SR}_A(\bar{x})$ is the least ordinal α such that for all $\bar{y} \in \mathrm{dom}(\mathcal{A})$, $\bar{x} \equiv_A^\alpha \bar{y}$ implies that $(\mathcal{A}, \bar{x}) \cong (\mathcal{A}, \bar{y})$; and similarly for $\mathcal{SR}_C(\bar{x})$. We first show that $\mathcal{SR}_\mathcal{A}(\bar{x}) \geq \mathcal{SR}_\mathcal{C}(\bar{x})$. Suppose $\mathcal{SR}_C(\bar{x}) = \beta$. We assume for a contradiction that $\mathcal{SR}_A(\bar{x}) = \gamma < \beta$. Consider an arbitrary $\bar{z} \in \Sigma^*$ (the domain of $\mathcal{C}$) such that $\bar{x} \equiv_\mathcal{C}^\gamma \bar{z}$. By Lemma 2.16, $\bar{x} \equiv_\mathcal{A}^\gamma \bar{z}$. But, the definition of γ as the Scott rank of $\bar{x}$ in $\mathcal{A}$ implies that $(\mathcal{A}, \bar{x}) \cong (\mathcal{A}, \bar{z})$. Now, $\mathcal{C}$ is $L_{\omega_1,\omega}$ definable in $\mathcal{A}$ and therefore inherits the isomorphism. Hence, $(\mathcal{C}, \bar{x}) \cong (\mathcal{C}, \bar{z})$. But, this implies that $\mathcal{SR}_\mathcal{C}(\bar{x}) \leq \gamma < \beta = \mathcal{SR}_\mathcal{C}(\bar{x})$, a contradiction.

The conclusion of the above argument is that for each $\bar{x} \in \Sigma^*$, $\mathcal{SR}_\mathcal{A}(\bar{x}) \geq \mathcal{SR}_\mathcal{C}(\bar{x})$. Hence, since $\mathrm{dom}(\mathcal{C}) \subset \mathrm{dom}(\mathcal{A})$,

$$\begin{aligned}\mathcal{SR}(\mathcal{A}) &= \sup\{\mathcal{SR}_\mathcal{A}(\bar{x}) + 1 : \bar{x} \in \mathrm{dom}(\mathcal{A})\} \\ &\geq \sup\{\mathcal{SR}_\mathcal{A}(\bar{x}) + 1 : \bar{x} \in \mathrm{dom}(\mathcal{C})\} \\ &\geq \sup\{\mathcal{SR}_\mathcal{C}(\bar{x}) + 1 : \bar{x} \in \mathrm{dom}(\mathcal{C})\} = \mathcal{SR}(C).\end{aligned}$$

In the other direction, we wish to show that $\mathcal{SR}(\mathcal{A}) \leq 2 + \mathcal{SR}(\mathcal{C})$. Suppose this is not the case. Then there is $\bar{x}\bar{x}'\bar{u} \in \mathcal{A}$ such that $\mathcal{SR}_\mathcal{A}(\bar{x}\bar{x}'\bar{u}) \geq 2 + \mathcal{SR}(\mathcal{C})$. By Lemma 2.17, there is $\bar{y} \in \Sigma^*$ such that $2 + \mathcal{SR}_\mathcal{C}(\bar{y}) \geq 2 + \mathcal{SR}(\mathcal{C})$, a contradiction. □

Recent work in the theory of computable structures has focussed on finding computable structures of high Scott rank. Nadel [77] proved that any computable structure has Scott rank at most $\omega_1^{CK} + 1$. Early on, Harrison [44] showed that

there is a computable ordering of type $\omega_1^{CK}(1+\eta)$ (where η is the order type of the rational numbers). This ordering has Scott rank $\omega_1^{CK}+1$, as witnessed by any element outside the initial ω_1^{CK} set. However, it was not until much more recently that a computable structure of Scott rank ω_1^{CK} was produced (see Knight and Millar [67]). A recent result of Cholak, Downey, and Harrington gives the first natural example of a structure with Scott rank ω_1^{CK}: the computably enumerable sets under inclusion [29].

Corollary 2.19. *There is an automatic structure with Scott rank ω_1^{CK}. There is an automatic structure with Scott rank $\omega_1^{CK}+1$.*

We also apply the construction to [41], where it is proved that there are computable structures with Scott ranks above each computable ordinal. In this case, we get the following theorem.

Theorem 2.20. *For each computable ordinal α, there is an automatic structure of Scott rank at least α.*

2.6 Cantor-Bendixson rank of automatic successor trees

In this section we show that there are automatic successor trees of high Cantor-Bendixson (CB) rank. Recall the definitions of partial order trees and successor trees from Section 2.1. Note that if $(T;\leq)$ is an automatic partial order tree then the associated successor tree $(T;S)$, where the relation S is defined by

$$S(x,y) \iff (x<y) \;\&\; \neg\exists z(x<z<y),$$

is automatic.

Definition 2.21. The **derivative** of a (partial order or successor) tree T, $d(T)$, is the subtree of T whose domain is

$$\{x \in T : \ x \text{ lies on at least two infinite paths in } T\}.$$

By induction, $d^0(T) = T$, $d^{\alpha+1}(T) = d(d^\alpha(T))$, and for γ a limit ordinal, $d^\gamma(T) = \cap_{\beta<\gamma} d^\beta(T)$. The **CB rank** of the tree, $CB(T)$, is the least α such that $d^\alpha(T) = d^{\alpha+1}(T)$.

The CB ranks of automatic partial order trees are finite [61]. We will show that this is not true of automatic successor trees. The main theorem of this section provides a general technique for building trees whose CB ranks are predetermined. Before we prove it, we give some examples of automatic successor trees with relatively low CB ranks.

Example 2.22. There is an automatic partial order tree (hence an automatic successor tree) whose CB rank is n for each $n \in \omega$.

Proof. The tree T_n is defined over the n letter alphabet $\{a_1, \ldots, a_n\}$ as follows. The domain of the tree is $a_1^* \cdots a_n^*$ (put $\lambda = a_1^0$). The order $\leq_n$ is the prefix partial order. Therefore, the successor relation is given as follows:

$$S(a_1^{\ell_1} \cdots a_i^{\ell_i}) = \begin{cases} \{a_1^{\ell_1} \cdots a_i^{\ell_i+1}, a_1^{\ell_1} \cdots a_i^{\ell_i} a_{i+1}\} & \text{if } 1 \leq i < n \\ \{a_1^{\ell_1} \cdots a_i^{\ell_i+1}\} & \text{if } i = n \end{cases}$$

For example, the tree in the case where $n = 2$ is given in Figure 2.1.

Note that if $n = 0$ then the tree is empty, which is consistent with it having CB rank 0. It is obvious that, for all n, T_n is an automatic partial order tree. The rank of T_n can be shown, by induction, to be equal to n. The base case is easy.

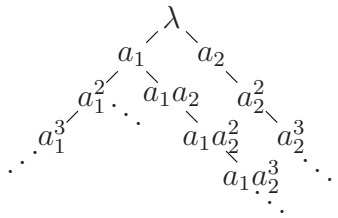

Figure 2.1: Automatic partial order tree with CB rank 2

Assume that $CB(T_n) = n$. Then T_{n+1} is the tree T_n where a single infinite branch is appended to each node. So $d(T_{n+1}) = T_n$ and $CB(T_{n+1}) = CB(T_n)+1 = n+1$. □

The following examples code the finite rank successor trees uniformly into one automatic tree so as to push the rank higher. Our first example is an automatic successor tree $T_{\omega+1}$ of rank $\omega + 1$. The following lemma from [63] informs our constructions:

Lemma 2.23 (Khoussainov, Rubin, Stephan; 2005)**.** *Suppose $\mathcal{T}$ is a countable tree and $CB(\mathcal{T}) = \alpha$. If $d^\alpha(\mathcal{T}) = \emptyset$ then $\mathcal{T}$ has at most countably many paths and $CB(\mathcal{T})$ is 0 or a successor ordinal. If $d^\alpha(\mathcal{T}) \neq \emptyset$ then $d^\alpha(\mathcal{T})$ and $\mathcal{T}$ both contain uncountably many infinite paths.*

Therefore, $T_{\omega+1}$ will have countably many paths. Later, we construct a tree of rank ω which must embed the perfect tree because its CB rank is a limit ordinal.

Example 2.24. There is an automatic successor tree $T_{\omega+1}$ whose CB rank is $\omega+1$.

Proof. Informally, this tree is a chain of trees of increasing finite CB ranks. Let

$T_{\omega+1} = (\{0,1\}^*; S)$ with S defined as follows:

$$\begin{cases} S(1^n) = \{1^n 0, 1^{n+1}\} & \text{for all } n \\ S(0u) = \{0u0\} & \text{for all } u \in \{0,1\}^* \\ S(1^n 0u) = \{1^n 0u0, 1^{n-1}0u1\} & \text{for } n \geq 1 \text{ and } u \in \{0,1\}^* \end{cases}$$

Intuitively, the subtree of rank n is coded by the set X_n of nodes which contain exactly n 1s. By induction on the length of strings, we can show that range$(S) = \{0,1\}^*$ and hence the domain of the tree is also $\{0,1\}^*$. The transitive closure of the relation S satisfies the conditions of being a tree. Each of the clauses of the successor relation is easy to recognise and hence $T_{\omega+1}$ is automatic. It remains to compute the rank of $T_{\omega+1}$. We note that in successive derivatives, each of the finite rank sub-trees X_n is reduced in rank by 1. Therefore $d^{\omega}(T_{\omega+1}) = 1^*$. But, since each point in 1^* is on exactly one infinite path, $d^{\omega+1}(T) = \emptyset$, and this is a fixed-point. Thus, $CB(T_{\omega+1}) = \omega + 1$, as required. □

The following example gives a tree T_ω of rank ω. The idea is to code the trees T_n into the leftmost path of the full binary tree.

Example 2.25. There is an automatic successor tree T_ω whose CB rank is ω.

Proof. The tree is the full binary tree, where at each node on the leftmost branch we append trees of increasing finite CB rank. Thus, define $T_\omega = (\{0,1\}^* \cup \{0,a\}^*; S)$ where S is given as follows:

$$\begin{cases} S(u1v) = \{u1v0, u1v1\} & \text{for all } u, v \in \{0,1\}^* \\ S(0^n) = \{0^{n+1}, 0^n 1, 0^n a\} & \text{for all } n \\ S(au) = \{aua\} & \text{for all } u \in \{0,a\}^* \\ S(0^n au) = \{0^n aua, 0^{n-1}au0\} & \text{for } n \geq 1 \text{ and } u \in \{0,a\}^* \end{cases}$$

Proving that T_ω is an automatic successor tree is a routine check. So, we need only compute its rank. Each derivative leaves the right part of the tree (the full binary tree) fixed. However, the trees appended to the leftmost path of the tree are affected by taking derivatives. Successive derivatives decrease the rank of the protruding finite rank trees by 1. Therefore, $d^\omega(T_\omega) = \{0,1\}^*$, a fixed point. Thus, $CB(T_\omega) = \omega$. □

To extend these examples to higher ordinals, we consider the **product** operation on trees defined as follows. Let $(T_1; S_1)$ and $(T_2; S_2)$ be successor trees and let r_1 be the root of $\mathcal{T}_1$, r_2 be the root of $\mathcal{T}_2$. The product of these trees is the tree $(T_1 \times T_2; S_\times)$ with successor relation given by:

$$S_\times((x,y),(u,v)) \iff \begin{cases} y = r_2 \ \& \ [(u = x, S_2(y,v)) \vee (S_1(x,u), y = v)] \\ y \neq r_2 \ \& \ [u = x, S_2(y,v)]. \end{cases}$$

Proposition 2.26. *Assume that $\mathcal{T}_1$ and $\mathcal{T}_2$ are successor trees of CB ranks α and β, respectively, each having at most countably many paths. Then $\mathcal{T}_1 \times \mathcal{T}_2$ has CB rank $\beta + \alpha$. Moreover, if $\mathcal{T}_1$ and $\mathcal{T}_2$ are automatic successor trees then so is the product.*

Proof. Since automatic structures are closed under first-order definitions, if $\mathcal{T}_1$ and $\mathcal{T}_2$ are automatic then so is $(T_1 \times T_2; S_\times)$. The definition of $S_\times$ also guarantees that the product structure is a successor tree. It remains to calculate its CB rank. Notice that the product tree can be thought of as $\mathcal{T}_1$ where a copy of $\mathcal{T}_2$ is attached to each node. Since $d^\beta(\mathcal{T}_2) = \emptyset$, $d^\beta(\mathcal{T}_1 \times \mathcal{T}_2) = \mathcal{T}_1$. Therefore, $d^{\beta+\alpha}(\mathcal{T}_1 \times \mathcal{T}_2) = d^\alpha \mathcal{T}_1 = \emptyset$ and the CB rank of the product tree is $\beta + \alpha$. □

The examples and the proposition above yield tools for building automatic successor trees of CB ranks up to ω^2. However, it is not clear that these methods

can be applied to obtain automatic successor trees of higher CB ranks. We will see that a different approach to building automatic successor trees will yield all possible CB ranks.

We are now ready for the main theorem of this section. As before, we will transfer results from computable trees to automatic trees. We note that every computable successor tree $(T; S)$ is also a computable partial order tree. Indeed, in order to effectively check whether $x \prec_S y$, we calculate the distances of y and x from the root. If y is closer to the root or is at the same distance as x then $\neg(x \prec_S y)$; otherwise, we start computing the trees above all z at the same distance from the root as x is. Since y is farther away from the root than x, it must appear in one of these trees. If it is in the subtree above x, we answer that $x \prec_S y$; if it appears in one of the other subtrees, $\neg(x \prec_S y)$. Thus, we have computed the partial order associated with S. Hence, every computable successor tree is a computable partial order tree. However, not every computable partial order tree is a computable successor tree. We have the following inclusions:

$$\text{Aut. PO trees} \subset \text{Aut. Succ. trees} \subset \text{Comp. Succ. trees} \subset \text{Comp. PO trees}$$

Using techniques similar to those of Examples 2.24 and 2.25, we can recursively code up computable trees of increasing CB rank. Hence, we have the following fact.

Fact 2.27. *For each $\alpha < \omega_1^{CK}$ there is a computable successor tree of CB rank α.*

Theorem 2.28. *For $\alpha < \omega_1^{CK}$ there is an automatic successor tree of CB rank α.*

Proof. Suppose we are given $\alpha < \omega_1^{CK}$. Take a computable tree R_α of CB rank α. We use the same construction as in the case of well-founded relations (see the proof of Theorem 2.10). The result is a stretched out version of the tree R_α,

where between each two elements of the original tree we have a coding of their computation. In addition, extending from each $x \in \Sigma^*$ we have infinitely many finite computation chains. Those chains which correspond to output "no" are not connected to any other part of the automatic structure. Finally, there is a disjoint part of the structure consisting of chains whose bases are not valid initial configurations. By the reversibility assumption, each unproductive component of the configuration space is isomorphic either to a finite chain or to an ω-chain. Moreover, the set of invalid initial configurations which are the base of such an unproductive chain is regular. We connect all such bases of unproductive chains to the root and get an automatic successor tree, T_α.

We now consider the CB rank of T_α. Note that the first derivative removes all the subtrees whose roots are at distance 1 from the root and are invalid initial computations. This occurs because each of the invalid computation chains has no branching and is not connected to any other element of the tree. Next, if we consider the subtree of T_α rooted at some $x \in \Sigma^*$, we see that all the paths which correspond to computations whose output is "no" vanish after the first derivative. Moreover, $x \in d(T_\alpha)$ if and only if $x \in d(R_\alpha)$ because the construction did not add any new infinite paths. Therefore, after one derivative, the structure is exactly a stretched out version of $d(R_\alpha)$. Likewise, for all $\beta < \alpha$, $d^\beta(T_\alpha)$ is a stretched out version of $d^\beta(R_\alpha)$. Hence, $CB(T_\alpha) = CB(R_\alpha) = \alpha$. □

Automatic successor trees have also been recently studied by Kuske and Lohrey in [69]. Techniques similar to those above are used to show that the existence of an infinite path in an automatic successor tree is Σ^1_1-complete. In addition, Kuske and Lohrey look at graph questions for automatic graphs and show that the existence of a Hamiltonian path is Σ^1_1-complete whereas the set cover problem is decidable.

2.7 Conclusion

In this chapter, we studied the complexity of automatic structures. In particular, we examined the differences in complexity between automatic and computable structures. We showed that automatic well-founded partial orders are considerably simpler than their computable counterparts, because the ordinal heights of automatic partial orders are bounded below ω^ω. On the other hand, computable well-founded relations, computable successor trees, and computable structures in general can be transformed into automatic objects in a way which (almost) preserves the ordinal height, Cantor-Bendixson rank, or Scott ranks. Therefore, the corresponding classes of automatic structures are as complicated as possible.

Chapter 3

Algorithmic Properties of Unary Automatic Structures

The results in this chapter were reported in [52]. This is joint work with Bakhadyr Khoussainov and Jiamou Liu.

3.1 Introduction

We study the algorithmic properties of infinite graphs that result from a natural unfolding operation applied to finite graphs. The unfolding process always produces infinite graphs of finite degree. Moreover, the class of resulting graphs is a subclass of the class of automatic graphs. As such, any member of this class possesses all the known algorithmic and algebraic properties of automatic structures. An equivalent way to describe these graphs employs automata over a unary alphabet (see Theorem 3.11). Therefore, we call this class of graphs **unary automatic graphs of finite degree**. Since these graphs are described by the unfolding operation (Definition 3.10) on a pair of finite graphs $(\mathcal{D}, \mathcal{F})$, this pair serves as a finite representation of the infinite graph. The size of this pair is the sum of the sizes of the deterministic automata that represent these graphs. We use this notion of size in our complexity considerations, noting that we require the finite graph presentations to be via deterministic unary automata. We are interested in the following natural decision problems:

- **Connectivity**: given an automatic graph $\mathcal{G}$, decide if $\mathcal{G}$ is connected.

- **Reachability**: given an automatic graph $\mathcal{G}$ and two vertices x and y of the graph, decide if there is a path from x to y.

If we restrict to the class of finite graphs, these two problems are decidable and can be solved in linear time on the sizes of the graphs. However, we are interested in infinite graphs and therefore much more work is needed to investigate the problems above. Moreover, the infinite graph context gives rise to the following additional problems:

- **Infinity Testing**: given an automatic graph $\mathcal{G}$ and a vertex x, decide if the component of $\mathcal{G}$ containing x is infinite.
- **Infinite Component**: given an automatic graph $\mathcal{G}$, decide if $\mathcal{G}$ has an infinite component.

For the full class of automatic graphs all of the above problems are undecidable. In fact, exact bounds on undecidability are known: the connectivity problem is Π^0_2-complete; the reachability problem is Σ^0_1-complete; the infinite component problem is Σ^0_3-complete; and the infinity testing problem is Π^0_2-complete [90].

Unary automatic structures are all first-order definable in $S1S$ (the monadic second-order logic of the successor function on the natural numbers). Hence, each of the problems above is decidable when restricted to the class of unary automatic structures [11], [90]. Direct constructions using this definability in $S1S$ yield algorithms with nonelementary time complexity since one needs to transform $S1S$ formulas into automata [22]. However, we provide polynomial-time algorithms for solving all the above problems for this class of graphs.

One might ask whether there is an intermediate class of automatic graphs between unary automatic graphs and arbitrary automatic graphs. The following example illustrates that, in the context of the graph problems we are considering, any intermediate class would be intractable. Let $\mathcal{G}_{1^*2^*}$ be the class of graphs whose set of nodes is $1^*2^* = \{1^n2^m : n, m \in \mathbb{N}\}$ and whose set of edges is finite automaton recognisable. (In this notation, unary automatic graphs may be denoted $\mathcal{G}_{1^*}$.) The infinite grid

$$G_2 = (\mathbb{N} \times \mathbb{N}; \{(\langle i, j\rangle, \langle i', j'\rangle : (i = i' \& j' = j_1) \ \vee \ (i' = i + 1 \& j = j')\})$$

is in $\mathcal{G}_{1^*2^*}$ by the encoding $(i, j) \mapsto 1^i2^j$. However, the monadic second order theory of G_2 is not decidable [101], and in particular, counter machines can be coded into the grid. Hence, the reachability problem for G_2 is not decidable. Thus, even in this simple extension of $\mathcal{G}_{1^*}$, the graph problems in question do not have algorithmic solutions.

We now outline the rest of this chapter by explaining the main results. Section 3.2 defines unary automatic graphs and provides a characterization theorem (Theorem 3.4) for them. Section 3.3 introduces unary automatic graphs of finite degree. The main result is Theorem 3.11 that explicitly provides an algorithm for building unary automatic graphs of finite degree. This theorem is used throughout the chapter. Section 3.4 is devoted to deciding the infinite component problem. The main result is the following:

Theorem 3.13 *The infinite component problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(n^3)$, where n is the number of states of the deterministic unary finite automaton recognising $\mathcal{G}$.*

In this section, we make use of the concept of oriented walk for finite directed graphs. Section 3.5 is devoted to deciding the infinity testing problem. The main

result is the following:

Theorem 3.16 *The infinity testing problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(n^3)$, where n is the number of states of the deterministic unary finite automaton $\mathcal{A}$ recognising $\mathcal{G}$. In particular, when $\mathcal{A}$ is fixed, there is a constant time algorithm that decides the infinity testing problem on $\mathcal{G}$.*

The fact that there is a constant time algorithm when $\mathcal{A}$ is fixed will be made clear in the proof. The value of the constant is polynomial in the number of states of $\mathcal{A}$.

The reachability problem is addressed in Section 3.6. This problem has been studied in [19], [40], [95] via the class of **pushdown graphs**. A pushdown graph is the configuration space of a pushdown automaton. In [95], Thomas shows that all unary automatic graphs are pushdown graphs. Given a pushdown graph $\mathcal{G}$ and a node v, there is a finite automaton that recognises all nodes reachable from v. The size of this automaton depends on the input node v (see [19],[40], [95]). Once such an automaton is produced, checking the reachability problem for nodes v and u amounts to running the automaton on u. This is an algorithm to decide the reachability probelm for any unary automatic graph of finite degree. For brevity, we will refer to this algorithm as the pushdown approach.

Theorem 3.19 gives an alternate solution to the reachability problem for unary automatic graphs of finite degrees that improves on the pushdown approach in several ways. In the pushdown approach, the finite automata constructed are not uniform in v: different automata are built for different input nodes v. The approach we present will be uniform in u and v. Moreover, the automata from the pushdown approach are nondeterministic. The equivalent deterministic finite

automata are then exponential in the size of the representation of v, and hence the pushdown approach uses exponential space. The algorithm presented in Section 3.6 constructs a deterministic automaton $\mathcal{A}_{Reach}$ that accepts the set of pairs $\{\langle u, v\rangle :$ there is a path from u to $v\}$. The size of $\mathcal{A}_{Reach}$ only depends on the number of states of the automaton recognising the edge relation of the graph $\mathcal{G}$; constructing the automaton requires polynomial time in the size of this edge automaton. The practical advantage of such a uniform solution is that once $\mathcal{A}_{Reach}$ is built, deciding whether node v is reachable from u takes only linear time (details are in Section 3.6). The main result of this section is the following:

Theorem 3.19 *Suppose $\mathcal{G}$ is a unary automatic graph of finite degree represented by deterministic unary finite automaton $\mathcal{A}$ of size n. There exists a polynomial-time algorithm that solves the reachability problem on $\mathcal{G}$. For inputs u, v, the running time of the algorithm is $O(|u| + |v| + n^4)$.*

Finally, Section 3.7 solves the connectivity problem for $\mathcal{G}$.

Theorem 3.25 *The connectivity problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(n^3)$, where n is the number of states of the deterministic unary finite automaton recognising $\mathcal{G}$.*

3.2 Unary automatic graphs

This section presents definitions and background for infinite graphs presentable by unary automata. We will assume throughout that the automata are deterministic. This assumption influences the algorithms and the complexity results discussed in this chapter. For a deterministic automaton, the following notion is well-defined.

Let q, q' be states in a given automaton. The **distance** from q to q' is the minimum number of transitions required for $\mathcal{A}$ to go from q_0 to q_1.

Definition 3.1. An infinite structure $\mathcal{A}$ is **unary automatic** if it has an automata presentation whose domain is 1^* and whose relations are automatic.

Examples of unary automatic structures are $(\omega; S)$ and $(\omega; \leq)$ (recall Section 1.2). The classes of unary automatic linearly ordered sets, permutation structures, graphs, and equivalence structures have well understood characterizations [60], [11]. For example, unary automatic linearly ordered sets are exactly those that are isomorphic to a finite sum of orders of type ω, ω^* (the order of negative integers), and finite n. In particular, the ordinals with unary automata presentations are exactly those below ω^2.

Definition 3.2. A **unary automatic graph** is a graph $(V; E)$ whose domain is 1^*, and whose edge relation E is a regular binary relation over 1^*.

In the rest of this chapter, we focus on deterministic unary automatic structures. Hence, we will use the word "automatic" to mean deterministic unary automatic. Moreover, all structures mentioned will be infinite unless explicitly specified otherwise. Finally, we will focus on undirected graphs. The case of directed graphs is an easy generalization of the methods discussed in this chapter, but involves bulky notation.

Let $\mathcal{G} = (V; E)$ be an automatic graph. Let $\mathcal{A}$ be an automaton recognising E. We establish some terminology for the automaton $\mathcal{A}$. The general shape of $\mathcal{A}$ is given in Figure 3.1. Note that since E is a binary relation recognised by a unary automaton, inputs to the automaton are finite words over $\{\binom{1}{1}, \binom{\diamond}{1}, \binom{1}{\diamond}\}$. All the states reachable from the initial state by reading inputs of type $\binom{1}{1}$ are called

$(1,1)$**-states**. A **tail** in $\mathcal{A}$ is a sequence of states linked by transitions without repetition. A **loop** is a sequence of states linked by transitions such that the last state coincides with the first one, and there are no other repetitions. The set of $(1,1)$-states is a disjoint union of a tail and a loop. We call the tail the $(1,1)$**-tail** and the loop the $(1,1)$**-loop**. Let s be a $(1,1)$-state. All the states reachable from s by reading inputs of type $\binom{1}{\diamond}$ are called $(1,\diamond)$**-states**. This collection of all $(1,\diamond)$-states is also a disjoint union of a tail and a loop, called the $(1,\diamond)$**-tail** and the $(1,\diamond)$**-loop**, respectively, of s. The $(\diamond,1)$**-tails** and $(\diamond,1)$**-loops** of $(1,1)$-states are defined in a similar matter.

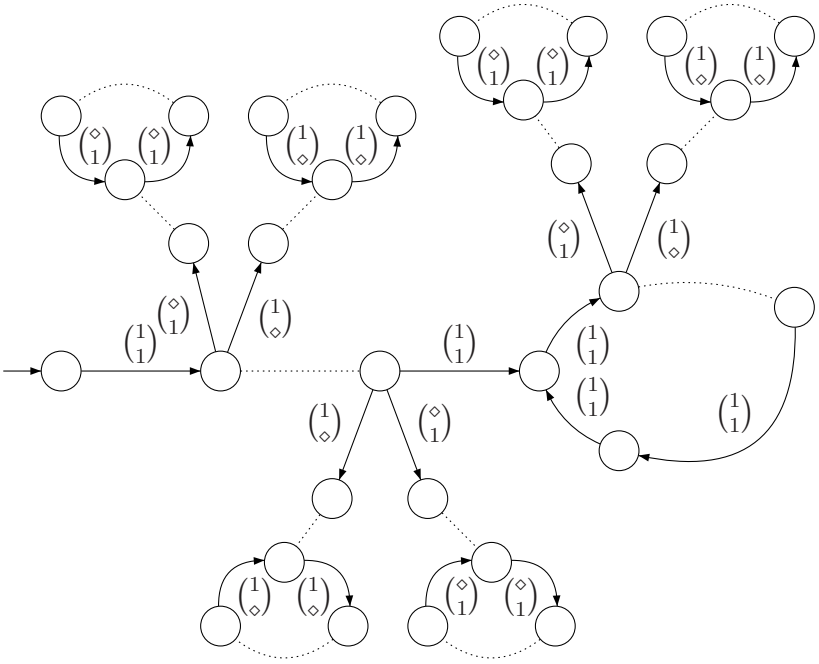

Figure 3.1: A typical unary graph automaton

We say that an automaton is **standard** if the lengths of all its loops and tails equal some number p, called the **loop constant**. If $\mathcal{A}$ is a standard automaton recognising 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. At the end of each $(1,\diamond)$-tail and $(\diamond,1)$-tail there is a $(1,\diamond)$-loop and $(\diamond,1)$-loop, respectively, of size

exactly p. Therefore, if n is the number of states in $\mathcal{A}$, then $n = 8p^2$.

Lemma 3.3. *Let $\mathcal{A}$ be an n state automaton recognising a binary relation E on 1^*. There exists an equivalent standard automaton with at most $8n^{2n}$ states.*

Proof. Let p be the least common multiple of the lengths of all loops and tails of $\mathcal{A}$. An easy estimate shows that p is no more than n^n. One can transform $\mathcal{A}$ into an equivalent standard automaton whose loop constant is p. Hence, there is a standard automaton equivalent to $\mathcal{A}$ whose size is bounded above by $8n^{2n}$. □

Our convention to restrict attention to undirected graphs simplifies the general shape of the automaton. Indeed, we need only consider transitions labelled by $\binom{\diamond}{1}$. To see this, given an automaton with only $\binom{\diamond}{1}$ transitions, to include all symmetric edges, add a copy of each $\binom{\diamond}{1}$ transition which is labelled with $\binom{1}{\diamond}$.

We recall a characterization theorem of unary automatic graphs from [90]. Let $\mathcal{B} = (B; E_B)$ and $\mathcal{D} = (D; E_D)$ be finite graphs. Let $R_1, R_2 \subset D \times B$, and $R_3, R_4 \subset B \times B$. Consider the graph $\mathcal{D}$ followed by countably infinitely many copies of $\mathcal{B}$, ordered as $\mathcal{B}^0, \mathcal{B}^1, \mathcal{B}^2, \ldots$. Formally, the vertex set of $\mathcal{B}^i$ is $B \times \{i\}$ and we write $b^i = (b, i)$ for $b \in B$ and $i \in \omega$. The edge set E^i of $\mathcal{B}^i$ consists of all pairs $\langle a^i, b^i \rangle$ such that $(a, b) \in E_B$. Using the finite graphs and the four binary relations as parameters we define the infinite graph, $unwind(\mathcal{B}, \mathcal{D}, \bar{R})$, as follows: the vertex set is $D \cup B^0 \cup B^1 \cup B^2 \cup \ldots$; the edge set contains $E_D \cup E^0 \cup E^1 \cup \ldots$ as well as the following edges, for all $a, b \in B$, $d \in D$, and $i, j \in \omega$:

- (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 $(a, b) \in R_4$.

Theorem 3.4 (Rubin; 2004)**.** *A graph is unary automatic if and only if it is isomorphic to unwind*$(\mathcal{B}, \mathcal{D}, \bar{R})$ *for some parameters* $\mathcal{B}$*,* $\mathcal{D}$*, and* $\bar{R}$*. Moreover, if* $\mathcal{A}$ *is a standard automaton representing* $\mathcal{G}$ *then the parameters* $\mathcal{B}, \mathcal{D}, \bar{R}$ *can be extracted in* $O(n^2)$*; otherwise, the parameters can be extracted in* $O(n^{2n})$*, where* n *is the number of states in* $\mathcal{A}$*.*

Another way of understanding unary automatic graphs is via their connection with **prefix-recognisable graphs**. There is a tradition in the literature (see [75], [25], and [76]) of studying classes of infinite graphs. In particular, Caucal [26] proposed a hierarchy of graphs $(\mathcal{G}_n)$ and trees $(\mathcal{T}_n)$ which can be generated from finite structures in a particular way. The main operation in going from the graph hierarchy to the tree hierarchy is **unfolding**, defined by Courcelle and Walukiewicz in [32]. The operation in the opposite direction is monadic second-order definability. A good overview of these transformations and the intuition behind Caucal's hierarchy may be found in [96]. It is known that each graph in Caucal's hierarchy has a decidable MSO theory. The low levels of this hierarchy are relevant to our discussion. The base of the tree hierarchy, $\mathcal{T}_0$, is the class of finite trees. The class $\mathcal{G}_0$ consists of the class of finite graphs. $\mathcal{T}_1$ consists of all regular trees (defined as infinite trees which have only finitely many nonisomorphic subtrees). For example, the infinite binary tree belongs to the class $\mathcal{T}_1$. The class $\mathcal{G}_1$ makes up an important class of graphs called the **prefix-recognisable graphs** [25], defined as follows.

Definition 3.5. Let $\mathcal{A}$ be a finite set and Σ be a finite alphabet. A graph is **prefix-recognisable** if it is isomorphic to a graph of the form $(S; (E_a)_{a \in A})$ where S is a regular language over Σ and each E_a is a finite union of relations of the form $W(V \times U) = \{(wu, wv) : u \in U, v \in V, w \in W\}$ for regular languages $U, V, W \subseteq \Sigma^*$.

In [12], Blumensath studies the connection between automatic graphs and prefix-recognisable graphs: the class of prefix-recognisable graphs is a proper subset of the class of automatic graphs, while the class of unary automatic graphs is a proper subset of the class of prefix-recognisable graphs. Therefore, the class of unary automatic graphs falls into $\mathcal{G}_1$ of Caucal's hierarchy.

Given a prefix-recognisable graph G, we say the **prefix-recognisable presentation** of G is a graph G_{pr} such that $G_{pr} \cong G$ and $G_{pr} = (S; (E_a)_{a \in A})$ as in Definition 3.5. Each prefix-recognisable presentation of G is given as a collection of automata recognising S and sets U, V, W for each E_a, respectively. To show that all unary automatic graphs are prefix-recognisable, [12] uses a MSO interpretation of unary automatic structures in the infinite binary tree. Here we present an alternative proof that directly constructs the prefix-recognisable presentation of a unary automatic graph.

Theorem 3.6. *Let G be a unary automatic graph represented by a standard automaton $\mathcal{A}$ with loop constant p. There is an algorithm that constructs its prefix-recognisable presentation G_{pr} in time $O(6p^3)$.*

Proof. In this proof, we use the notation from the definition of $unwind(\mathcal{B}, \mathcal{D}, \bar{R})$. From $\mathcal{A}$ we can compute the parameters $\mathcal{B}, \mathcal{D}, \bar{R}$ of G in time $O(p^2)$. Note that the number of vertices in each of $\mathcal{B}$ and $\mathcal{D}$ is p. Let $\Sigma = \{0, 1, 2\}$. We define the set S and sets of edges E_d, E_b, E_1, E_2, E_3, E_4 as follows:

- $S = \{02^m : m < p\} \cup \{02^{p-1}(01^{p-1})^*01^n : n < p\}$
- $E_d = \{(02^m, 02^n) : (m, n) \in E_D\}$
- $E_b = \{(02^{p-1}(01^{p-1})^i01^m, 02^{p-1}(01^{p-1})^i01^n) : (m, n) \in E_B, i \geq 0\}$
- $E_1 = \{(02^m, 02^{p-1}01^n : (m, n) \in R_1\}$

- $E_2 = \{(02^m, 02^{p-1}(01^{p-1})^i 01^n) : (m,n) \in R_2, i > 0\}$
- $E_3 = \{(02^{p-1}(01^{p-1})^i 01^m, 02^{p-1}(01^{p-1})^{i+1} 01^n) : (m,n) \in R_3, i \geq 0\}$
- $E_4 = \{(02^{p-1}(01^{p-1})^i 01^m, 02^{p-1}(01^{p-1})^{i+j} 01^n : (m,n) \in R_4, i \geq 0, j > 1\}$

The set of vertices in our graph G_{pr} is S, and the set of edges is the union of all the edge relations defined above. By the construction, the finite graph $\mathcal{D}$ is represented by the set $\{02^m : m < p\}$, and for each $i \geq 0$, the i^{th} copy of $\mathcal{B}$ is represented by $\{02^{p-1}(01^{p-1})^i 01^n : n < p\}$. The edge sets E_d and E_b respectively represent E_D and E_B, and for $1 \leq i \leq 4$, E_i represents R_i.

It is easy to see that $G \cong G_{pr}$. We need to show that G_{pr} is prefix-recognisable. By definition, S is a regular language. Notice also that each edge in $E_D \cup E_B \cup \bar{R}$ defines a relation in G_{pr} of the form $W(U \times V)$ where W, U, V are regular languages over Σ. Since E_D, E_B and $\bar{R}$ are finite sets, the edge relation of G_{pr} can be written as a finite union of relations of the above form.

For each of the regular languages above, the automaton recognising the language has $O(p)$ transitions. Since the total number of edges in $E_D \cup E_B \cup \bar{R}$ is bounded above by $6p^2$, we need to construct at most $6p^2$ automata. Therefore the total running time to construct G_{pr} is $O(6p^3)$. □

3.3 Unary automatic graphs of finite degree

A graph is of **finite degree** if there are at most finitely many edges at each vertex. We call a unary automaton recognising a binary relation a **one-loop automaton** if its transition diagram contains exactly one loop, the $(1,1)$-loop. Note that if a loop has no accepting states then it is irrelevant to the language of the automaton

and we treat it as though it were missing. The general structure of one-loop automata is given in Figure 3.2.

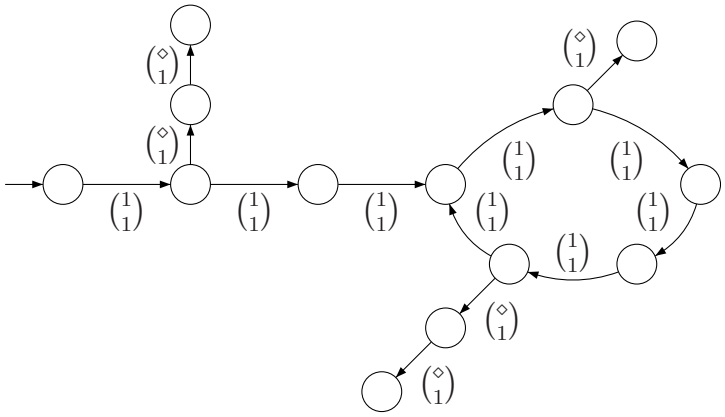

Figure 3.2: A typical one-loop automaton

We will always assume that the lengths of all the tails of the one-loop automata are not bigger than the size of the $(1,1)$-loop.

Proposition 3.7. *Let $\mathcal{G} = (V; E)$ be a unary automatic graph, then $\mathcal{G}$ is of finite degree if and only if there is a one-loop automaton $\mathcal{A}$ recognising E.*

Proof. Suppose that $\mathcal{A}$ is not a one-loop automaton. Then there is a state q on the $(1,1)$-tail or $(1,1)$-loop of $\mathcal{A}$ which has a $(\diamond, 1)$-loop. Let i be the distance between the initial state of $\mathcal{A}$ and q; let j be the distance between q and the first accepting state on the $(\diamond, 1)$-loop from q; and, let k be the size of q's $(\diamond, 1)$-loop. Then the tuples $\langle 1^i, 1^{i+j+nk} \rangle$ (for any n) are all accepted by the automaton, and hence $\mathcal{G}$ is not of finite degree. The converse is similar. □

By Lemma 3.3, transforming a given automaton to an equivalent standard automaton may increase the number of states exponentially. However, there is

only polynomial blow up if $\mathcal{A}$ is a one-loop automaton.

Lemma 3.8. *If $\mathcal{A}$ is a one-loop automaton with n states, there exists an equivalent standard one-loop automaton with loop constant $p \leq n$.*

Proof. Let l be the length of the loop in $\mathcal{A}$ and t be the length of the longest tail in $\mathcal{A}$. Define p to be the least multiple of l such that $p \geq t$. It is easy to see that $p \leq l + t \leq n$. One can transform $\mathcal{A}$ into an equivalent standard one-loop automaton whose loop constant is p. □

Note that the equivalent standard automaton has $2p$ (1,1)-states. From each of them there is a $(1, \diamond)$-tail of length p and a $(\diamond, 1)$-tail of length p. Hence the automaton has $4p^2$ states. By the above lemma, we always assume the input automaton $\mathcal{A}$ is standard. In the rest of the chapter, we will state all results in terms of the loop constant p 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 $\mathcal{G}_1 = (1^*; E_1)$ and $\mathcal{G}_2 = (1^*; E_2)$, we can form the **union graph** $\mathcal{G}_1 \oplus \mathcal{G}_2 = (1^*; E_1 \cup E_2)$ and the **intersection graph** $\mathcal{G}_1 \otimes \mathcal{G}_2 = (1^*; E_1 \cap E_2)$. Automatic graphs of finite degree are closed under these operations. Indeed, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be one-loop automata recognising E_1 and E_2 with loop constants p_1 and p_2, respectively. The usual product construction for building union and intersection automata produces a one-loop automaton whose loop constant is $p_1 \cdot p_2$. We introduce another operation: consider the new graph $\mathcal{G}_1' = (1^*; E_1')$ where the set E_1' of edges is defined as follows: a pair $\langle 1^n, 1^m \rangle$ is in E' if and only if $\langle 1^n, 1^m \rangle \notin E$ and $|n - m| \leq p_1$. The relation E_1' is recognised by the same automaton as E_1, modified so that all $(\diamond, 1)$-states that are final declared

non-final, and all the $(\diamond, 1)$-states that are non-final declared final. Thus, we have the following proposition:

Proposition 3.9. *If $\mathcal{G}_1$ and $\mathcal{G}_2$ are unary automatic graphs of finite degree then so are $\mathcal{G}_1 \oplus \mathcal{G}_2$, $\mathcal{G}_1 \otimes \mathcal{G}_2$, and $\mathcal{G}_1'$.* □

Now our goal is to recast Theorem 3.4 for graphs of finite degree. Our analysis will show that, in contrast to the general case for automatic graphs, the parameters $\mathcal{B}$, $\mathcal{D}$, and $\bar{R}$ for graphs of finite degree can be extracted in linear time.

Definition 3.10 (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 set $\Sigma_{\mathcal{D},\mathcal{F}}$ consisting of all mappings $\eta : V_\mathcal{D} \to \mathcal{P}(V_\mathcal{F})$, and the finite set $\Sigma_\mathcal{F}$ consisting of all mappings $\sigma : V_\mathcal{F} \to \mathcal{P}(V_\mathcal{F})$. An 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, defines the infinite graph $\mathcal{G}_\alpha = (V_\alpha; E_\alpha)$ as follows:

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

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

Figure 3.3 illustrates the general shape of a unary automatic graph of finite degree that is built from $\mathcal{D}$, $\mathcal{F}$, η, and σ^ω, where σ^ω is the infinite word $\sigma\sigma\sigma\cdots$.

Theorem 3.11. *A graph of finite degree $\mathcal{G} = (V; E)$ has a unary automata 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 $\mathcal{G}_{\eta\sigma^\omega}$.*

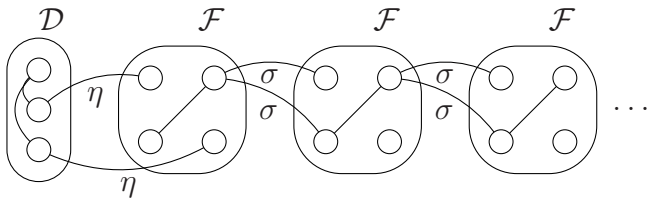

Figure 3.3: Unary automatic graph of finite degree $\mathcal{G}_{\eta\sigma^\omega}$

Proof. Let $\mathcal{G} = (V; E)$ be a unary automatic graph of finite degree and $\mathcal{A}$ be an automaton recognising E. In linear time in the number of states of $\mathcal{A}$, $\mathcal{A}$ can be transformed into a one-loop automaton. So, assume without loss of generality that $\mathcal{A}$ is a one-loop automaton with loop constant p. Label the states of $\mathcal{A}$ as follows: q_0 is the starting state, $q_1, \ldots, q_{p-1}$ are successive states of the $(1,1)$-tail. Define the finite graph $\mathcal{D}$ by $V_\mathcal{D} = \{q_0, q_1, \ldots, q_{p-1}\}$, and for $0 \leq i \leq j < p$, $(q_i, q_j) \in E_\mathcal{D}$ if and only if 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 $j - i$. Similarly, define $\mathcal{F}$ by $V_\mathcal{F} = \{q'_0, \ldots, q'_{p-1}\}$ where $q'_0, \ldots, q'_{p-1}$ are all states on the $(1,1)$-loop, and the edge relation $E_\mathcal{F}$ is analogous to $E_\mathcal{D}$. For $m, n \in \{0, \ldots, p-1\}$, put q'_n in $\eta(q_m)$ if and only if there exists a final state q_f in the $(\diamond, 1)$-tail of q_m, and the distance from q_m to q_f is $p + n - m$. 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, suppose we are given a graph $\mathcal{G}_{\eta\sigma^\omega}$. Relabel $V_\mathcal{D} = \{q_0, \ldots, q_{\ell-1}\}$, $V_\mathcal{F} = \{q'_0, \ldots, q'_{p-1}\}$. A one-loop automaton $\mathcal{A}$ recognising the edge relation of $\mathcal{G}_{\eta\sigma^\omega}$ is constructed as follows. The $(1,1)$-tail of the automaton is $\{q_0, \ldots, q_{\ell-1}\}$ and the $(1,1)$-loop is $\{q'_0, \ldots, q'_{p-1}\}$, 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, it is easy to 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 3.12. *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, $|V_\mathcal{F}| = |V_\mathcal{D}| = p$.* □

3.4 Deciding the infinite component problem

We continue the convention that any input graphs we consider are undirected. A **component** of a graph is the transitive closure of a vertex under the edge relation. The **infinite component problem** asks whether a given graph has an infinite component.

Theorem 3.13. *The infinite component 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 recognising $\mathcal{G}$.*

By Theorem 3.11, let $\mathcal{G} = \mathcal{G}_{\eta\sigma^\omega}$. We assume without loss of generality that $\mathcal{D} = \emptyset$ and $\mathcal{G} = \mathcal{G}_{\sigma^\omega}$, 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 vertex x in $\mathcal{F}^i$. We will define a finite *directed* graph $\mathcal{F}^\sigma$ as an auxiliary tool in our analysis of the *undirected* input graph $\mathcal{G}$. Let $\mathcal{F}^\sigma = (V^\sigma; E^\sigma)$ be defined as follows. Each node in V^σ represents a distinct connected component of $\mathcal{F}$. 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 the pair $\langle x, y\rangle \in E^\sigma$ if and only if $y' \in \sigma(x')$ for some x' and y' that are in the same components as x and y, respectively. Constructing $\mathcal{F}^\sigma$ requires finding connected components of $\mathcal{F}$ and hence takes time $O(p^2)$. To prove Theorem 3.13, we will make essential use of the following definition, which is discussed in [45].

Definition 3.14. An **oriented walk** in a *directed* graph $\mathcal{H}$ is a subgraph $\mathcal{P}$ of $\mathcal{H}$ that consists of a sequence of nodes $v_0, ..., v_k$ such that for $1 \leq i \leq k$, either $\langle v_{i-1}, v_i\rangle$ or $\langle v_i, v_{i-1}\rangle$ is an edge in $\mathcal{H}$, and for each $0 \leq i \leq k$, exactly one of $\langle v_{i-1}, v_i\rangle$ and $\langle v_i, v_{i-1}\rangle$ 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 $\langle v_i, v_{i+1}\rangle$ is called a **forward arc** and $\langle v_{i+1}, v_i\rangle$ is called a **backward arc**. The **net length** of $\mathcal{P}$, denoted $disp(\mathcal{P})$, is the difference between the number of forward arcs and the number of backward arcs. Note that the net length can be negative. The next lemma establishes a connection between oriented cycles in $\mathcal{F}^\sigma$ and infinite components of $\mathcal{G}$.

Lemma 3.15. *$\mathcal{G}$ contains an infinite component if and only if there is an oriented cycle in $\mathcal{F}^\sigma$ whose net length 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 vertices in the set $\{x^{jm+i} : j \in \omega\}$ 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 C 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 C.

Let x^i and x^j be two copies of x in C 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 vertex $y \in V_{\mathcal{F}}$ 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 3.13. By the equivalence in Lemma 3.15, it suffices to give an algorithm that decides if $\mathcal{F}^\sigma$ contains an oriented cycle with positive net length. But, the existence of an oriented cycle with positive net length is equivalent to the existence of an oriented cycle with negative net length. Therefore, Algorithm 3.1 finds oriented cycles with nonzero 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 labelled queue of nodes Q_x which are connected to x.

At each point in Algorithm 3.1 where we are building a queue for node x and processing z, both $d(z)$ and $d'(z)$ represent net lengths of paths from x to z. We claim that the algorithm returns *YES* if and only if there is an oriented cycle in $\mathcal{F}^\sigma$ with nonzero net length. Suppose Algorithm 3.1 returns *YES*. 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 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 nonzero net length, then it is our required oriented cycle; otherwise, we disregard the part between the

Algorithm 3.1: `Oriented-Cycle`

1: **while** there is node $x \in F^\sigma$ for which a queue has not been built, **do**
2: Initialize queue Q_x as empty.
3: Put $d(x) = 0$, put x into Q_x, mark x as *unprocessed*.
4: **for all** y *unprocessed* in Q_x **do**
5: **for all** $z \in \{z : \langle y, z\rangle \in E^\sigma \ \vee \ \langle z, y\rangle \in E^\sigma\}$ **do**
6: **if** $\langle y, z\rangle \in E^\sigma$ **then**
7: Set $d'(z) = d(y) + 1$.
8: **if** $z \notin Q_x$ **then**
9: Set $d(z) = d'(z)$, put z into Q_x, mark z as *unprocessed*.
10: **else**
11: **if** $d(z) \neq d'(z)$ **then**
12: **return** *YES*
13: **if** $\langle z, y\rangle \in E^\sigma$ **then**
14: Set $d'(z) = d(y) - 1$.
15: **if** $z \notin Q_x$ **then**
16: Set $d(z) = d'(z)$, put z into Q_x, mark z as *unprocessed*.
17: **else**
18: **if** $d(z) \neq d'(z)$ **then**
19: **return** *YES*
20: Mark y as *processed*.
21: **return** *NO*

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 nonzero net length where $x_0 = x_m$. We assume for a contradiction that Algorithm 3.1 returns *NO*. Consider how the algorithm acts when we pick x_0 at step (1). For each $0 \leq i \leq m$, one can prove the following statements by induction on i.

$(*)$ x_i always gets a label $d(x_i)$

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

By the description of Algorithm 3.1, x_0 gets the label $d(x_0) = 0$. Suppose $(*), (**)$ hold for x_i, $0 \leq i < m$. At the next stage of the algorithm, the algorithm labels all nodes in $\{z : \langle z, x_i \rangle \in E^\sigma \text{ or } \langle x_i, z \rangle \in E^\sigma\}$. 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} already has a label $d(x_{i+1})$ and $d(x_{i+1}) \neq d'(x_{i+1})$, then the algorithm would return *YES*. 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 Algorithm 3.1 is correct.

To summarize, the following algorithm solves the infinite component problem. Let $\mathcal{A}$ be a given automaton (with loop constant p) which recognises the unary automatic graph $\mathcal{G}$ of finite degree. Recall that p is also the cardinality of $V_\mathcal{F}$. We first compute $\mathcal{F}^\sigma$, in time $O(p^2)$. We then run `Oriented-Cycle` 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 $\mathcal{F}^\sigma$ has p nodes, this takes time $O(p^3)$.

Note that Lemma 3.15 holds even when $|V_\mathcal{F}| > |\mathcal{F}/\sim_{comp}|$. Therefore, a slight

modification to the algorithm above solves this case as well. □

3.5 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 vertex v and a graph $\mathcal{G}$, decides if v belongs to an infinite component of $\mathcal{G}$. We prove the following theorem.

Theorem 3.16. *The infinity testing problem for unary automatic graphs $\mathcal{G}$ of finite degree and input vertex v is solved in $O(p^3)$, where p is the loop constant of the automaton $\mathcal{A}$ recognising $\mathcal{G}$. In particular, when $\mathcal{A}$ is fixed, there is a constant time algorithm that decides the infinity testing problem on $\mathcal{G}$ for any given input vertex v.*

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

Lemma 3.17. *If x^i is connected to some y^j such that $|j-i| > p$, then x^i is in an infinite component of $\mathcal{G}$.*

Proof. Suppose such a y^j exists. Let P be a path from x^i to y^j in $\mathcal{G}$. Since $p = |V_{\mathcal{F}}|$, 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 $\{z^{s+(t-s)m} : m \in \omega\}$ are in the same component as x^i. □

Let $i' = min\{p, i\}$. To decide if 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 vertices in $\mathcal{F}^{i-i'}, \ldots, \mathcal{F}^{i+p}$. The algorithm is presented as Algorithm 3.2.

Algorithm 3.2: `FiniteReach`

1: Initialize the queue Q to be empty. Put the pair $(x, 0)$ into Q and mark it as *unprocessed*.
2: **for all** *unprocessed* pairs $(y, d) \in Q$ **do**
3: **for all** edges $e = \langle y, z\rangle$ or $e = \langle z, y\rangle$ in E^σ **do**
4: **if** $e = \langle y, z\rangle$ **then**
5: Set $d' = d + 1$.
6: **else**
7: Set $d' = d - 1$.
8: **if** $-i' \leq d' \leq p$ and (z, d') is not in Q **then**
9: Put (z, d') into Q and mark (z, d') as *unprocessed*
10: Mark (y, d) as *processed*.

Thus, 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 `FiniteReach` on the input 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` is bounded by the number of edges in $\mathcal{G}$ restricted to $\mathcal{F}^0, \ldots, \mathcal{F}^{2p}$. Therefore the running time of `FiniteReach` is $O(p^3)$. Define $B_{x^i} = \{y : (y, p) \in Q$ after running `FiniteReach` on $x^i\}$.

Lemma 3.18. *Let $x \in V_{\mathcal{F}}$. x^i is in an infinite component if and only if $B_{x^i} \neq \emptyset$.*

Proof. Suppose there is some $y \in B_{x^i}$. Then there is a path from x^i to y^{i+p} and, by Lemma 3.17, x^i is in an infinite component. Conversely, if x^i is in an infinite component, there must be a vertex in $\mathcal{F}^{i+p}$ reachable from x^i. Let y^{i+p} be such a vertex. There must be a path from x^i to y^{i+p} such that y^{i+p} is the first vertex in

$\mathcal{F}^{i+p}$ appearing on this path. Then $y \in B_{x^i}$. □

Proof of Theorem 3.16. We assume the input vertex x^i is presented as the pair (x, i). The above lemma suggests a simple algorithm to check if x^i is in an infinite component.

Algorithm 3.3: `InfiniteTest`

1: Run `FiniteReach` on vertex x^i, computing B_{x^i} while building Q.
2: **for all** $y \in B_{x^i}$ **do**
3: **if** there is some edge $\langle y, z\rangle \in E^\sigma$ **then**
4: **return** *YES*
5: **return** *NO*

Running `FiniteReach` takes $O(p^3)$ and checking for edge $\langle y, z\rangle$ takes $O(p^2)$. The running time is therefore $O(p^3)$. If the graph and its presentation $\mathcal{A}$ are fixed and we are given any input vertex $v = x^i$, since x is bounded by p checking whether x^i belongs to an infinite component takes constant time. □

3.6 Deciding the reachability problem

The **reachability problem** on a graph $\mathcal{G}$ is: given two vertices x^i, y^j in $\mathcal{G}$, decide if x^i and y^j are in the same component. We prove the following theorem.

Theorem 3.19. *Suppose $\mathcal{G}$ is a unary automatic graph of finite degree represented by an automaton $\mathcal{A}$ with loop constant p. There exists a polynomial-time algorithm that solves the reachability problem on $\mathcal{G}$. For inputs u and v, the running time of the algorithm is $O(|u| + |v| + p^4)$.*

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

Since, by Theorem 3.16, there is an $O(p^3)$-time algorithm to check if x^i is in a finite component, we can work on the two possible cases separately. We first deal with the case in which x^i is in a finite component. By Lemma 3.17, x^i and y^j are in the same (finite) component if and only if after running `FiniteReach` on the input x^i, the pair $(y, j-i)$ is in the queue Q.

Corollary 3.20. *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 deciding if 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 vertices $y \in V_{\mathcal{F}}$ whose i^{th} copy lies in the same $\mathcal{G}$-component as x^i. The algorithm is similar to `FiniteReach`, except that it does not depend on the input i. Lines 8 to 9 in Algorithm 3.2 are changed to the following:

---

if $-p \leq d' \leq p$ and (z, d') is not in Q **then**

  Put (z, d') into Q and mark (z, d') as *unprocessed*

We use this modified algorithm to define the set $Reach(x) = \{y : (y, 0) \in Q\}$. 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, $y \in Reach(x)$ 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 3.21. *Suppose x^i is in an infinite component. The vertex y^i is in the same component as x^i if and only if y^i is also in an infinite component and $y \in Reach(x)$.*

Proof. Suppose y^i is in an infinite component and $y \in Reach(x)$. 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 $y \in Reach(x)$, 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 vertices in the set $\{x^{p+rm} : m \in \omega\}$ are in the same component. Likewise, there is an $r' > 0$ such that all vertices in $\{y^{p+r'm} : m \in \omega\}$ 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 $P_1'P_2'$ 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 $y \in Reach(x)$. 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, $y \in Reach(x)$. 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 to satisfy the assumption. Let z be a vertex 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 vertices in the sequence $(x^i, s^j, s^{k+j}, z^p, t^{m+n}, t^m, y^i)$. Consider the following sequence of vertices:

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

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

We can extend the definition of *Reach* and σ to subsets of $V_{\mathcal{F}}$ by taking the the union of $Reach(x)$ or $\sigma(x)$ for each x in the subset. We inductively define a sequence $Cl_0(x), Cl_1(x), \ldots$ such that each $Cl_k(x)$ is a subset of $V_{\mathcal{F}}$. Let $Cl_0(x) = Reach(x)$ and for $k > 0$, we define $Cl_k(x) = Reach(\sigma(Cl_{k-1}(x)))$. The following lemma is immediate from this definition.

Lemma 3.22. *Suppose x^i is in an infinite component. Then x^i and y^j are in the same component of $\mathcal{G}$ if and only if y^j is also in an infinite component and $y \in Cl_{j-i}(x)$.* □

We can use Lemma 3.22 to define a naïve algorithm that solves the reachability problem on inputs x^i, y^j; this is Algorithm 3.4.

Algorithm 3.4: `NaïveReach`

if x^i is in an infinite component of $\mathcal{G}$ **then**
  if y^j is in an infinite component of $\mathcal{G}$ **then**
    Compute $Cl_{j-i}(x)$.
    if $y \in Cl_{j-i}(x)$ **then**
      return *YES*
    else
      return *NO*
  else
    return *NO*
else if y^j is in a finite component of $\mathcal{G}$ **then**
  Run `FiniteReach` on input x^i.
  if $(y, j-i) \in Q$ **then**
    return *YES*
  else
    return *NO*
else
  return *NO*

The worst case complexity of Algorithm 3.4 is as follows. The set $Cl_0(x)$ can be computed in time $O(p^3)$. Given $Cl_{k-1}(x)$, we can compute $Cl_k(x)$ in time $O(p^3)$ by computing $Reach(y)$ for any $y \in \sigma(Cl_{k-1}(x))$. Therefore, the total running time of `NaïveReach` on input x^i, y^j is $(j-i) \cdot p^3$. We want to replace the multiplication with addition.

From Lemma 3.18, x^i is in an infinite component in $\mathcal{G}$ if and only if `FiniteReach` finds a vertex y^{i+p} connecting to x^i. Now, suppose that x^i is in an infinite compo-

nent. We can use `FiniteReach` to find such a y and a path from x^i to y^{i+p}. On this path, there must be two vertices 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 vertices in the set $\{x^{i+mr} : m \in \omega\}$ belong to the same component.

Lemma 3.23. *$Cl_0(x) = Cl_r(x)$.*

Proof. By definition, $y \in Cl_0(x)$ 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 $y \in Cl_r(x)$. For the reverse inclusion, suppose $y \in Cl_r(x)$. 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, `Reach`, on inputs x^i, y^j by replacing lines 3 to 7 from Algorithm 3.4 with

---

if x^i and y^j belong to infinite components **then**
  Compute $Cl_0(x), \ldots, Cl_{r-1}(x)$.
  if there is $k < r$ such that $j - i = k \mod r$ and $y \in Cl_k(x)$ **then**
    return *YES*
  else
    return *NO*

Proof of Theorem 3.19. Suppose the input vertices are x^i and y^j. By Lemma 3.22 and Lemma 3.23, the algorithm `Reach` returns *YES* if and only if x^i and y^j are in the same component. Since $r \leq p$, calculating $Cl_0(x), \ldots, Cl_{r-1}(x)$ requires time $O(p^4)$. Therefore the running time of `Reach` on input x^i, y^j is $O(i + j + p^4)$. □

In fact, the algorithm produces a number $k < p$ such that in order to check if x^i and y^j $(j > i)$ are in the same component, we need to test if $j-i < p$ and if $j-i = k \mod p$. Therefore if $\mathcal{G}$ is fixed and we compute $Cl_0(x), \ldots, Cl_{r_x-1}(x)$ for all x beforehand, then deciding whether two vertices 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 3.24. *Given a unary automatic graph $\mathcal{G}$ of finite degree represented by an automaton with loop constant p, there is a deterministic automaton with at most $2p^4 + p^3$ states that solves the reachability problem on $\mathcal{G}$. The time required to construct this automaton is $O(p^5)$.*

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

1. $\mathcal{A}_{Reach}$ has a $(1,1)$-tail of length p^2. Label the initial state by q_0 and the states on the tail by $q_0, q_1, \ldots, q_{p^2-1}$. These states represent vertices 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 vertices 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 vertices 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 rp 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 vertices 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 $\hat{q}_x^1, \ldots, \hat{q}_x^{p^2}$ and represent vertices 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 rp. We write these states as $\tilde{q}_x^1, \ldots, \tilde{q}_x^{rp}$.

The final (accepting) states of $\mathcal{A}_{Reach}$ 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 `FiniteReach` on input 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 $Cl_0(x), \ldots, Cl_{r-1}(x)$.

   (a) Make state q_{ip+x}^{jp+y-x} final if y^{i+j} is in an infinite component and $y \in Cl_j(x)$.

   (b) Make state $\check{q}_{ip+x}^{jp+y-x}$ final if $y \in Cl_j(x)$

4. If x^p is in a finite component, run the algorithm `FiniteReach` on input 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 $Cl_0(x), \ldots, Cl_{r-1}(x)$. Declare state $\tilde{q}_x^{jp+y-x}$ final if $y \in Cl_j(x)$.

One can show that $\mathcal{A}_{Reach}$ is the desired automaton. To compute the complexity of building $\mathcal{A}_{Reach}$, 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 3.16.

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 `FiniteReach` on input x^i. This takes time $O(p^5)$ by Corollary 3.20.

3. For all $x \in V_{\mathcal{F}}$ such that x^p is in an infinite component, compute the sets $Cl_0(x), \ldots, Cl_{r-1}(x)$. This requires time $O(p^5)$ by Theorem 3.19.

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

3.7 Deciding the connectivity problem

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

Algorithm 3.5: `NaïveConnect`

1: Construct the automaton $\mathcal{A}_{Reach}$.
2: **if** all states in $\mathcal{A}_{Reach}$ are accepting states **then**
3: **return** *YES*
4: **else**
5: **return** *NO*

`NaïveConnect` runs in time $O(p^5)$. However, the structural properties of finite degree unary automatic graphs suggest that a more intuitive algorithm might be used to efficiently solve the connectivity problem. Indeed, this is the case.

Theorem 3.25. *Given a unary automatic graph $\mathcal{G}$ of finite degree, we can check if it is connected in time $O(p^3)$, where p is the loop constant of the automaton*

recognising the graph.

Observe that if $\mathcal{G}$ is infinite (as we assume it to be) but does not contain an infinite component, then $\mathcal{G}$ is not connected. This can be checked in $O(p^3)$ by Theorem 3.13. Therefore, we assume that $\mathcal{G}$ contains an infinite component C.

Lemma 3.26. *For all $i \in \mathbb{N}$, there is a vertex in $\mathcal{F}^i$ belonging to C.*

Proof. Since C is infinite, there is a vertex x^i and $s > 0$ such that all vertices in $\{x^{i+ms} : m \in \omega\}$ 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 vertex 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` on x^0 to compute the queue Q. Set $R = \{y \in V_{\mathcal{F}} : (y, 0) \in Q\}$.

Lemma 3.27. *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 vertex $y \in V_{\mathcal{F}} - R$. 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 3.21, 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 $\{y \in V_{\mathcal{F}} : (y, i) \in Q\}$ for $i \geq 0$ equals $V_{\mathcal{F}}$. By Lemma 3.26, all vertices are in the same component. □

Proof of Theorem 3.25. By Lemma 3.27, Algorithm 3.6 decides the connectivity problem on G.

Algorithm 3.6: `Connectivity`

1: **if** algorithm from Theorem 3.13 says that $\mathcal{G}$ contains an infinite component **then**
2: Pick $x \in V_{\mathcal{F}}$ and run `FiniteReach` on x^0. Let $C = \{y : (y, 0) \in Q\}$
3: **if** $C = V_{\mathcal{F}}$ **then**
4: **return** *YES*
5: **else**
6: **return** *NO*
7: **else**
8: **return** *NO*

Solving the infinite component problem takes $O(p^3)$ by Theorem 3.13. Running algorithm `FiniteReach` also takes $O(p^3)$. Therefore `Connectivity` takes $O(p^3)$.

□

3.8 Conclusion

In this chapter we addressed algorithmic problems for graphs of finite degree that have automata presentations over a unary alphabet. We provided polynomial-time algorithms that solve connectivity, reachability, infinity testing, and infinite component problems. There are are many other algorithmic problems for finite graphs that can be studied for the class of unary automatic graphs. These, for example, may include finding spanning trees for automatic graphs, studying the isomorphism problems, and other related issues.

Chapter 4

Automatic Decision Procedures

This chapter discusses decision procedures for various logical theories using automata theoretic methods. Recently, such decision procedures have led to applications in computer science. In particular, we will see how decision procedures associated with automatic structures give rise to alternate solutions for linear programming problems. Linear programming involves finding optimal solutions to systems of constraint equations. If the solutions are required to have integer values, linear programming is used in discrete optimization problems and control theory, in modern compilers for such tasks as dependence analysis for loop transformation [84], and for verification of hardware design [21], [28]. Moreover, if variables may range over either real numbers or the integers we have applications in hybrid systems and timed systems. In each of these settings, linear programming corresponds to a fragment of the first-order theory of a particular logical structure. We examine this theory more closely, and look at decision procedures via quantifier elimination and automata. While many of the results presented in Sections 4.1 and 4.2 have previously appeared, we provide an exposition of them as a coherent whole. Moreover, some of the theorems have thus far only been published as parts of extended abstracts in which major proofs were omitted; for completeness, we provide these proofs. The automata approach is extendible to other natural theories. In particular, Sections 4.3 and 4.4 present new decision procedures for the first-order theories of the valued field of the p-adic numbers and formal Laurent series with coefficients in finite fields, equipped with a valuation.

4.1 ILP and Presburger arithmetic

Linear programming problems involve the optimization of a linear quantity based on a system of linear equations defining the feasible domain of solutions. This abstraction arose from many disparate problems in transportation routing, supply chain, and military applications (to name a few). In 1939, Kantorovich first defined linear programming in the context of optimizing the allocation of resources. Later but independently, in 1947, the common features of many such problems were isolated and Dantzig began in earnest the study of linear programming as a subject. More recently, linear programming has been applied extensively in compiler design. To optimize the use of data stored in cache, loop transformations may be automatically applied to high level code. However, checking if such loop transformations preserve the semantics of the code can be formulated as a linear programming problem; if they are legal, finding the transformed loop bounds can be done via the enumeration of solutions of a linear programming problem.

A linear programming problem is specified as the simultaneous solution of a system of linear equations and inequalities in a way which minimizes or maximizes a linear cost function. The standard form of the problem consists of the linear objective (or cost) function $\mathbf{c} \cdot \mathbf{x}$, the system of inequalities $\mathbf{Ax} \leq \mathbf{b}$, and non-negative variables $\mathbf{x} \geq 0$. If the solution is required to be integer valued, the problem is an instance of **Integer Linear Programming** (ILP).

Fourier-Motzkin elimination of variables [33] was the original method for solving systems of linear equations. An extension of Fourier-Motzkin for integers has been implemented as the Omega test [84]. Given a system of linear equations and inequalities over integer variables, the method first uses Gaussian elimination and the greatest-common-denominator test to get solutions for the equations (if they

exist). If there are no solutions, then the system is unsatisfiable. However, if there are solutions to the equations in the system, the variables in the inequalities are parametrized by these solutions, and Fourier-Motzkin elimination is used to project each variable out of the remaining inequalities. The projected variable is expressed in terms of maxima and minima involving expressions in the still-free variables. At the end, we remain with inequalities involving a single variable and integers. This is easy to simplify, and then ripple the effect to all the variables.

The Simplex method, invented by Dantzig in the late 1940s, takes a more geometric view of the problem. Each linear inequality in an ILP problem defines a half-space of $\mathbb{Z}^n$. Therefore, the feasibility region is a convex polytope in $\mathbb{Z}^n$. It is not hard to see that if optimal solutions exist, they are at vertices of this region. The Simplex method looks for optimal solutions among these vertices. Both the Simplex method and the Omega test are conservative: they may claim a solution exists to a system of equations even if there is no actual integer solution.

In recent years, much work has been done on linear programming and ILP. In particular, the ellipsoid method in the late 1970s showed that linear programming (over the real numbers) has worst-case polynomial-time complexity as compared to ILP which is NP-complete. In the 1980s, interior point methods began emerging and are still actively studied. In the following, we examine ILP both from logical and automata theoretic perspectives.

Recall from Example 1.11 that Presburger arithmetic is the first-order theory of $(\mathbb{N}; +, \leq, 0, 1)$. Note that in some practical applications, Presburger arithmetic is taken to be the first-order theory of $(\mathbb{Z}; +, \leq, 0, 1)$. However, this amounts to a simple reduction and does not affect decidability or complexity results. The atomic formulas are linear equations and inequalities of the form $a_1x_1 + \cdots + a_nx_n =$

c or $a_1x_1 + \cdots + a_nx_n \leq c$, where $\langle a_1, \ldots, a_n \rangle \in \mathbb{Z}^n$ and $c \in \mathbb{Z}$. Note that scalar multiplication is used as a notational abbreviation for a fixed number of iterations of the addition function. Similarly, the relations $\geqslant, >, <$ are all definable in Presburger arithmetic. Hence, the system of constraints in an ILP problem can be defined in the quantifier-free fragment of Presburger arithmetic. The existence of an optimal value for the objective function can be tested by deciding a first-order sentence of Presburger arithmetic. Moreover, a witness to this sentence solves the ILP problem. Thus, a decision procedure for Presburger arithmetic yields a solution of any ILP problem. Conversely, one can use ILP to formulate a decision procedure for the quantifier-free fragment of Presburger arithmetic (see Jaroslaw [50] and Brinkmann and Drechsler [21]). Each quantifier-free formula is translated to disjunctive normal form and each disjunct is tested for satisfiability using ILP.

In 1927, Presburger [83] gave a quantifier elimination decision procedure for the full first-order theory of this arithmetic. The algorithm was improved by Cooper [31] and by Reddy and Loveland in 1978 [86] for formulas with bounded quantifier alternations. Büchi [22] began the tradition of translating arithmetic statements to automata. Boudet and Comon [20] gave a more efficient translation, which was further improved by Wolper and Boigelot in [102], [103]. As Theorems 4.1 and 4.5 and Algorithms 4.1 and 4.2, we present this last algorithm, a slight modification of which was implemented in [17]. To apply the automata translation to compiler applications of ILP, the set of solutions of a formula need to be enumerated. We demonstrate how to do this at the end of this section. In Sections 4.2, 4.3, and 4.4, we will see that this automata approach is amenable to extension for more complicated theories.

Given a formula $\varphi(x_1, \ldots, x_n)$ in Presburger arithmetic, we will construct a

finite automaton A_φ accepting the set $\{\langle x_1, \ldots, x_n\rangle \in \mathbb{Z}^n : (x_1, \ldots, x_n) \vDash \varphi\}$. Questions about the satisifiability of the formula and its witnesses (as in the ILP context) will then be easy to answer using automata theoretic techniques. We encode integers in base 2 representation with most significant bit first, and using 2's complement for negative numbers. Hence, a word $b_1 \cdots b_n$ encodes the value $-b_1 2^{n-1} + b_2 2^{n-2} + \cdots + b_{n-1} 2^1 + b_n$. A given formula φ may have multiple free variables. Correspondingly, solutions will be vectors of integers. We represent such vectors as vectors of encoded integers, all of which have equal length (we repeat the sign bit in the encoding of each component of the vector as many times as necessary to ensure encodings of equal length). The associated automaton $\mathcal{A}_\varphi$ will be synchronous and have as many input tapes as the number of free variables in φ. Moreover, we require that arbitrary repetitions of the sign bit in the encoding of integers will not affect whether the input vector is accepted or rejected by $\mathcal{A}_\varphi$. The following method generalizes easily to any fixed base r.

Base case, Part 1: Automata for Equations. Let φ be $a_1 x_1 + \cdots + a_n x_n = c$ for $\mathbf{a} = \langle a_1, \ldots, a_n\rangle \in \mathbb{Z}^n$ and $c \in \mathbb{Z}$. Let $\mathbf{a} \cdot \mathbf{x}$ abbreviate $a_1 x_1 + \cdots + a_n x_n$; φ may be written as $\mathbf{a} \cdot \mathbf{x} = c$. Each state of $\mathcal{A}_\varphi$ will represent the value of $\mathbf{a} \cdot \mathbf{x}$ derived from the currently known value of $\mathbf{x}$. Transitions between states will then reflect the contribution of the most recently learned bit of $\mathbf{x}$ to the computation. Thus, let $\mathcal{A}^1_\varphi = (\mathbb{Z} \cup \{\iota\}, \iota, \delta^1, \{c\})$, where

$$\delta^1(s, \mathbf{b}) = \begin{cases} -\mathbf{a} \cdot \mathbf{b} & \text{if } s = \iota \\ 2s + \mathbf{a} \cdot \mathbf{b} & \text{if } s \in \mathbb{Z} \end{cases}$$

The transition function uses the encoding of integers: after the first bit, we transition in a way which amounts to shifting the current value of the computation to the left and adding the newest input bit. Observe that $\mathcal{A}_\varphi$ is deterministic. The following theorem from [102] which proves the connection between the language

of $\mathcal{A}^1_\varphi$ and tuples satisfying φ appeared in an extended abstract without explicit proof.

Theorem 4.1 (Wolper, Boigelot; 1995)**.** *For* $\mathbf{w} \in \Sigma^*$*,* $\mathbf{w} \in L(\mathcal{A}^1_\varphi)$ *if and only if* $\mathbf{w}$ *encodes a solution* $\mathbf{x}$ *to* $\varphi \equiv \mathbf{a} \cdot \mathbf{x} = c$*.*

Proof. We prove the stronger statement: for $\mathbf{w} \in \Sigma^*$ and $s \in \mathbb{Z} \cup \{\iota\}$, there is a run of $\mathcal{A}^1_\varphi$ on $\mathbf{w}$ ending at state s if and only if $\mathbf{w}$ encodes a solution $\mathbf{x}$ to the equation $\mathbf{a} \cdot \mathbf{x} = s$. We go by induction on $|\mathbf{w}|$. Suppose $\mathbf{w} = \sigma_{\mathbf{0}}$, encoding the vector $\mathbf{x} = \langle -\sigma_{0,1}, \ldots, -\sigma_{0,n} \rangle$. Then the run of $\mathcal{A}^1_\varphi$ is $s_i s_1$ where $s_1 = -\mathbf{a} \cdot \sigma_{\mathbf{0}}$. Thus, $s_1 = -\mathbf{a} \cdot \langle -\sigma_{0,1}, \ldots, -\sigma_{0,n} \rangle = \mathbf{a} \cdot \mathbf{x}$, as required. For the inductive step, suppose that for any encoding $\sigma_{\mathbf{0}} \ldots \sigma_{\mathbf{m}}$ of $\mathbf{y}$, $\mathbf{y}$ satisfies $\mathbf{a} \cdot \mathbf{y} = s$ if and only if the run of $\mathcal{A}^1_\varphi$ on $\sigma_{\mathbf{0}} \ldots \sigma_{\mathbf{m}}$ ends at s. Let $\mathbf{w} = \sigma_{\mathbf{0}} \ldots \sigma_{\mathbf{m}} \sigma_{\mathbf{m+1}}$. Then $\mathbf{w}$ encodes the vector $\mathbf{x} = 2\mathbf{y} + \sigma_{\mathbf{m+1}}$. The run of $\mathcal{A}^1_\varphi$ on $\mathbf{w}$ is the result of appending the state s' to the run on $\sigma_{\mathbf{0}} \ldots \sigma_{\mathbf{m}}$, where by the definition of the transition function,

$$s' = 2s + \mathbf{a} \cdot \sigma_{\mathbf{m+1}} = 2(\mathbf{a} \cdot \mathbf{y}) + \mathbf{a} \cdot \sigma_{\mathbf{m+1}} = \mathbf{a} \cdot (2\mathbf{y} + \sigma_{\mathbf{m+1}}) = \mathbf{a} \cdot \mathbf{x}$$

as required. □

The definition of $\mathcal{A}^1_\varphi$ does not yield a finite automaton: the state space of $\mathcal{A}^1_\varphi$ is an infinite set. However, the accepting state s is not reachable from any state whose absolute value is sufficiently large. Therefore, all such states can be pruned from $\mathcal{A}^1_\varphi$ to yield a true finite automaton. Formally, let $||\mathbf{a}||_1 = \sum_{i=1}^n |a_i|$ and suppose $|s| > \max(|c|, ||\mathbf{a}||_1)$. Then note that for any $\mathbf{b} \in \Sigma$,

$$|2s + \mathbf{a} \cdot \mathbf{b}| \leq 2|s| + |\mathbf{a} \cdot \mathbf{b}| \leq 2|s| + ||\mathbf{a}||_1.$$

Therefore, any transition from state s satisfying the above conditions leads to a state s' such that $|s'| > |c|$. In turn, s' satisfies the same magnitude conditions as

s and hence transitions from it also move away (in absolute value) from c. Thus, for any state s such that $|s| > \max(|c|, ||\mathbf{a}||_1)$, there is no path from s to c. In light of this, define $\mathcal{A}^2_\varphi$ to be the automaton whose states are $(\mathbb{Z} \cap \{s : |s| \leq \max(|c|, ||\mathbf{a}||_1)\}) \cup \{\iota, s_+, s_-\}$ and whose transition function is

$$\delta^2(s, \mathbf{b}) = \begin{cases} -\mathbf{a} \cdot \mathbf{b} & \text{if } s = \iota, \\ s' & \text{if } s' = 2s + \mathbf{a} \cdot \mathbf{b}, s' \leq \max(|c|, ||\mathbf{a}||_1), \\ s_+ & \text{if } 2s + \mathbf{a} \cdot \mathbf{b} > \max(|c|, ||\mathbf{a}||_1), \\ s_- & \text{if } 2s + \mathbf{a} \cdot \mathbf{b} < -\max(|c|, ||\mathbf{a}||_1). \end{cases}$$

Note that a version of Theorem 4.1 where $\mathcal{A}^1_\varphi$ is replaced by $\mathcal{A}^2_\varphi$ holds because modifying the transition function as above does not affect which words are accepted by the automaton. The finite automaton A^2_φ is deterministic but is not minimal. To eliminate those states from which the accepting state is not reachable, we construct the automaton backwards and only include the necessary states. Thus, we arrive at Algorithm 4.1 from [103].

Let $\mathcal{A}_\varphi$ be the automaton whose states are given by the value of H at the end of this algorithm, initial state ι, $F = \{c\}$ and whose transitions are given by D. The underlying graph of $\mathcal{A}_\varphi$ is a subgraph of that of $\mathcal{A}^2_\varphi$ and hence $\mathcal{A}_\varphi$ is deterministic. Moreover, $L(\mathcal{A}_\varphi) = L(\mathcal{A}^2_\varphi)$ and $\mathcal{A}^2_\varphi$ is a minimal complete automaton.

Base case, Part 2: Automata for Inequalities. Let ψ be $a_1x_1 + \cdots + a_nx_n \leq c$ for $\mathbf{a} = \langle a_1, \ldots, a_n \rangle \in \mathbb{Z}^n$ and $c \in \mathbb{Z}$. As before, ψ may be written as $\mathbf{a} \cdot \mathbf{x} \leq c$. Similar reasoning as in the equation case applies here. Thus, we define

$$\mathcal{A}^1_\psi = \left((\mathbb{Z} \cap \{s : |s| \leq \max(|c|, ||\mathbf{a}||_1)\}) \cup \{\iota, s_+, s_-\}, s_i, \delta'^1, \{c\}\right),$$

where

$$\delta'^1 = \delta^2 \cup \{(s, \mathbf{b}, s') : \text{ there is some } s'' \leq s \text{ such that } \delta^2(s, \mathbf{b}) = s''\}.$$

Algorithm 4.1: `MinAutomatonEquation`

1: Initialize a list of states H to $\{\iota, c, s_+, s_-\}$; initialize a list of active states L to $\{c\}$; create a table D for transitions.
2: **while** $L \neq \emptyset$ **do**
3: Remove a state s from L.
4: **for all** $\mathbf{b} \in \{0,1\}^n$ **do**
5: **if** $\frac{s-\mathbf{a}\cdot\mathbf{b}}{2} \in \mathbb{Z}$ and $s' = \frac{s-\mathbf{a}\cdot\mathbf{b}}{2}$ is not already in H **then**
6: Add s' to H and L
7: Add a transition labelled by $\mathbf{b}$ from s' to s to D.
8: **if** $s = -\mathbf{a}\cdot\mathbf{b}$ **then**
9: Add a transition labelled by $\mathbf{b}$ from s_i to s to D.

Thus, $\mathcal{A}^1_\psi$ has the same set of states, initial state, and accepting states as the automaton $\mathcal{A}_\varphi$ if φ is an equation with the same coefficients as ψ. However, $\mathcal{A}^1_\psi$ has more transitions. As an aside, note that if $\mathcal{A}^2_\psi$ is defined to have domain $\{s \in \mathbb{Z} : |s| \leq \max(|c|, ||\mathbf{a}||_1)\} \cup \{\iota, s_+, s_-\}$, initial state s_i, transition function δ^2, and accepting set $\{s \in S : s \in \mathbb{Z} \text{ and } s \leq c\}$ then $L(\mathcal{A}^1_\psi) = L(\mathcal{A}^2_\psi)$. In the following, we will be composing automata end-to-end so $\mathcal{A}^1_\psi$ will be more useful than $\mathcal{A}^2_\psi$ because it has only one accepting state.

It is possible to prune the automaton $\mathcal{A}^1_\psi$ as we did in the case of automata representing equations. The algorithm in this case is Algorithm 4.2.

Let $\mathcal{A}_\psi$ be the automaton with states H, initial state ι, accepting state c, and transitions $D \cup \{(s, \mathbf{b}, s') : \text{there is some } s'' \leq s \text{ such that } (s, \mathbf{b}, s'') \in D\}$. Note that $\mathcal{A}_\psi$ is not deterministic. However, deterministic automata are much more efficient to implement and manipulate. In general, determinizing a finite

Algorithm 4.2: `MinAutomatonInequality`

1: Initialize a list of states H to $\{\iota, c, s_+, s_-\}$; initialize a list of active states L to $\{c\}$; create a table D for transitions.
2: **while** $L \neq \emptyset$ **do**
3: Remove a state s from L.
4: **for all** $\mathbf{b} \in \{0,1\}^n$ **do**
5: **if** $\lfloor \frac{s - \mathbf{a}\cdot\mathbf{b}}{2} \rfloor$ is not already in H **then**
6: Add $s' = \lfloor \frac{s - \mathbf{a}\cdot\mathbf{b}}{2} \rfloor$ to H and L
7: Add a transition labelled by $\mathbf{b}$ from s' to s to D.
8: **if** $s = -\mathbf{a}\cdot\mathbf{b}$ **then**
9: Add a transition labelled by $\mathbf{b}$ from s_i to s to D.

automaton comes at an exponential cost in the size of the state space. However, the automata corresponding to linear inequalities are of a particular kind.

Definition 4.2 (Wolper, Boigelot; 2000)**.** Given a non-deterministic finite automaton $\mathcal{A} = (S, \iota, \delta, F)$, for each $s \in S$ let $\mathcal{A}_s = (S, s, \delta, F)$. Then $\mathcal{A}$ is said to be **ordered** if there is a constant-time decidable strict total order $\prec$ on S (or $S \setminus \{\iota\}$) such that if $s_1 \prec s_2$, then $L(\mathcal{A}_{s_1}) \subsetneq L(\mathcal{A}_{s_2})$.

Theorem 4.3 (Wolper, Boigelot; 2000)**.** *There is a linear-time algorithm which, from a given non-deterministic ordered finite automaton, finds an equivalent deterministic automaton with the same number of states.*

Proof. Without loss of generality, we can remove transitions from a state on a given input to all but the $\prec$-greatest one, since the set of words accepted from all other states is a subset of those accepted from it. Since this algorithm merely removes transitions at each state, it takes linear time in the size of the automaton.

Moreover, it produces a deterministic finite automaton with the same set of states as the original non-deterministic finite automaton. □

Lemma 4.4. *The finite automaton $\mathcal{A}_\psi$ defined above is ordered under $s_1 \prec s_2 \iff s_1 > s_2$ (where $<$ is the order on the integers).*

Proof. This follows from the interpretation of the states as labels for the right-hand side of an inequality whose left hand side is $\mathbf{a} \cdot \mathbf{x}$ and analysing the relationship of words starting at p, q for $p < q$. □

Let $\delta'^{det}(s, \mathbf{b}) = \min\{\delta^{in}(s, \mathbf{b})\}$ and define $\mathcal{A}_\psi^{det}$ be the finite automaton obtained from $\mathcal{A}_\psi$ by replacing the transition relation by δ'^{det}. Then $\mathcal{A}_\psi^{det}$ is a deterministic finite automaton and the following theorem is proved.

Theorem 4.5 (Wolper, Boigelot; 2000)**.** *$L(\mathcal{A}_\psi^{det})$ is all binary encodings of vectors in $\mathbb{Z}^n$ satisfying ψ.* □

Inductive step. Given a general formula of Presburger arithmetic, we will build an automaton whose language is all binary encodings of vectors of integers satisfying the formulas. We use standard constructions for Boolean operations of languages of deterministic finite automata. Suppose $\xi \equiv \neg\varphi$, where φ is a formula for which we have already built a finite automaton, $\mathcal{A}_\varphi$. The formulas ξ and φ have the same free variables. Assume without loss of generality that they each have n free variables. Then

$$\{\mathbf{w} \in (\{0,1\}^n)^* : \mathbf{w} \text{ encodes a vector satisfying } \psi\} =$$

$$(\{0,1\}^n)^* \setminus \{\mathbf{w} \in (\{0,1\}^n)^* : \mathbf{w} \text{ does not encode a vector satisfying } \psi\}$$

since any word over $\{0,1\}^n$ encodes some vector of integers. Hence, $\mathcal{A}_\xi$ is the complement automaton of $\mathcal{A}_\varphi$. By similar arguments, if $\xi \equiv \varphi \wedge \psi$ let $\mathcal{A}_\xi =$

$\mathcal{A}_\varphi \cap \mathcal{A}_\psi$, and if $\xi \equiv \varphi \vee \psi$ let $\mathcal{A}_\xi = \mathcal{A}_\varphi \cup \mathcal{A}_\psi$. Thus, we obtain finite automata corresponding to any quantifier-free formula of Presburger arithmetic. This is sufficient to represent any ILP system.

The connection between automata and formulas of Presburger arithmetic can be extended to include quantified formulas. This work parallels the original automata decision procedures of Büchi [22], [23] and Rabin [85] which were described in general by Khoussainov and Nerode [56]. Suppose $\xi \equiv \exists x_i \varphi$, assume that φ has n free variables and let $\mathcal{A}_\varphi$ be the finite automaton corresponding to φ. We would like to define an automaton over $\{0,1\}^{(n-1)}$ whose language is

$$\{\mathbf{w} \in \left(\{0,1\}^{(n-1)}\right)^* : (\forall 0 \leq j < n)(\exists \sigma_j \in \{0,1\}) \text{ such that if } \mathbf{w}' \text{ is defined}$$
$$\text{by inserting } \sigma_j \text{ in } i^{th} \text{ position of component } j \text{ of } \mathbf{w}, \text{ it is in } L(\mathcal{A}_\psi)\}$$

But, this is (almost) exactly the projection operation applied to finite automata (see Section 1.2). The slight modification we must make is that the finite automaton resulting from projection may no longer accept all encodings of vectors satisfying φ. To illustrate this, consider the case where $\langle 10, 2\rangle$ is encoded in the language of $\mathcal{A}_\psi$, as $\langle 0^m 01010, 0^m 00010\rangle$. After projecting out the first variable, only encodings of 2 with at least three leading zeroes are included in the language. To ensure that the automata we build for existential formulas accepts *any* encoding (as required), we can apply the following modification to the projection construction: for each $\mathbf{b} \in \{0,1\}^{(n-1)}$, add to $\delta(s_i, \mathbf{b})$ any states which are reachable from s_i by $\mathbf{b}^k$ for some finite k.

Note that the resulting finite automaton is non-deterministic. Hence, in order for the induction to go through (since our algorithm for complementation requires that the finite automaton be deterministic), we must determinize the resulting finite automaton. In the setting of quantified formulas, the trick with ordered

automata no longer works and we incur exponential growth in the size of the set of states. For universally quantified formulas, we first convert to the equivalent existential formula ($\forall x\ \varphi \equiv \neg\exists x(\neg\varphi)$) and then apply automata transformations as above.

The connection between first-order logic and automata can be further exploited in applications of ILP. To our knowledge, this is the first time these algorithms have appeared in the literature. One example is in model checking. If ILP or Presburger arithmetic is used to formulate model checking problems, we need to decide if a particular formula is satisfiable in a given structure. To do this, we construct the finite automaton corresponding to the formula and ask whether the language of the automaton is empty. Recall that the emptiness question is decidable in linear time in the number of states of the automaton. However, the state space of the automaton constructed for a given formula may have exponential blow up. Hence, satisfiability corresponds to non-emptiness of a particular automaton. Moreover, if $L(\mathcal{A}_\varphi)$ is not empty, the algorithm for checking this gives an example of a word accepted $\mathcal{A}_\varphi$. A quick translation from the binary encoding give a vector of integers satisfying φ. This witness may be useful in debugging. Another example is in the context of compiler optimizations. When a system of linear equations or inequalities is used to generate loop bounds for loop nests under affine loop transformations, all solutions to the system must be enumerated. Given $A\mathbf{i} \leq \mathbf{b}$, a description of the original loop bounds, and an invertible linear transformation $\mathbf{u} = T\mathbf{i}$ on the indices of the loops, we get the system: $AT^{-1}\mathbf{u} \leq \mathbf{b}$. This is a system of linear inequalities. Bounds on the values of components of U correspond to bounds for the transformed loop nests. We can formulate this question in Presburger arithmetic. Suppose that

$$\varphi(x_1, \ldots, x_n) \equiv (a_{1,1}x_1 + \cdots + a_{1,n}x_n \leq b_1) \wedge \cdots \wedge (a_{m,1}x_1 + \cdots + a_{m,n}x_n \leq b_m),$$

abbreviated as $\varphi(\mathbf{x}) \equiv A\mathbf{x} \leq \mathbf{b}$. To get bounds on x_1, we define the formulas

$$\psi_1(x_1) \equiv \exists x_2 \ldots x_n(\varphi(x_1, \ldots, x_n) \wedge \forall y(\varphi(y, x_2, \ldots, x_n) \rightarrow x_1 \leq y)),$$

$$\psi_1'(x_1) \equiv \exists x_2 \ldots x_n(\varphi(x_1, \ldots, x_n) \wedge \forall y(\varphi(y, x_2, \ldots, x_n) \rightarrow y \leq x_1)).$$

Any value satisfying ψ_1 (respectively ψ_1') is the minimum (respectively maximum) value of x_1 which might satisfy φ. By the translation to automata discussed above, we can build $\mathcal{A}_{\psi_1}$, $\mathcal{A}_{\psi_1'}$ and check for satisfiability. If they are satisfied, then we explicitly get (from the accepting path through the automaton) bounds for x_1. Let m_1, M_1 be these bounds. For any subsequent variable, we define similar formulas ψ_k, ψ_k' with parameters $m_1, \ldots m_{k-1}, M_1, \ldots, M_{k-1}$:

$$\begin{aligned}
\psi_k(x_k) \equiv & \exists x_{k+1} \ldots x_n \forall x_1 \ldots x_{k-1}([m_1 \leq x_1 \leq M_1 \wedge \cdots \wedge m_{k-1} \leq x_{k-1} \leq M_{k-1}] \rightarrow \\
& [\varphi(x_1, \ldots, x_n) \wedge \forall y(\varphi(x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots, x_n) \rightarrow x_k \leq y)]) \\
\psi_k'(x_k) \equiv & \exists x_{k+1} \ldots x_n \forall x_1 \ldots x_{k-1}([m_1 \leq x_1 \leq M_1 \wedge \cdots \wedge m_{k-1} \leq x_{k-1} \leq M_{k-1}] \rightarrow \\
& [\varphi(x_1, \ldots, x_n) \wedge \forall y(\varphi(x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots, x_n) \rightarrow y \leq x_k)])
\end{aligned}$$

Again, we get formulas in one free variable that represent the bounds on the current variable. Once these formulas are converted to equivalent automata, we can quickly find an encoding for this bound, and hence its value. Thus, the ease with which the automata formulation handles quantified statements allows it to generate bounds for vectors satisfying constraint equations.

The size of the automaton $\mathcal{A}_\varphi$ built for a formula φ of Presburger arithmetic is crucial for the performance of this approach to solving ILP problems. The existence of solutions of systems of equations is checked by answering the emptiness question for such an $\mathcal{A}_\varphi$, and this is done in linear time in the size of $\mathcal{A}_\varphi$. Wolper and Boigelot [103] give a nonelementary (unbounded tower of exponentials) upper bound on the size of the finite automaton generated by a Presburger formula. More

recently, Klaedtke [65] proposed some optimizations to the translation algorithm in [103] and proved that the tight worst-case upper bound on the size of such finite automata for Presburger formulas is triply exponential. The improved algorithm was implemented in LIRA [7]. It is worth mentioning that automata techniques for ILP and compiler optimizations allow calculations involving arbitrarily large numbers with full precision.

4.2 MILP and $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$

Mixed integer linear programming (MILP) is a framework in which we have constraint equations with variables that may vary over the real numbers or over the integers. As mentioned earlier, these types of systems are used to represent hybrid systems (in which a continuous system is interacting with a discrete program, inducing both continuous and logical constraints). The projection and elimination (Fourier-Motzkin) method has recently been extended to the mixed integer case [9]. This approach uses a predicate to indicate whether a variable ranges over integers only and then combines the Fourier-Motzkin and Omega tests to project variables out one at a time. One implementation of this algorithm is integrated into the Cooperating Validity Checker [35]. This particular implementation includes proof-production: if the checker claims there is no solution to a particular system, it gives a proof of this claim which can be externally verified. Such proofs help in identifying false positive replies of the decision procedure, but still do not give an example of a solution.

Analogous to the role Presburger arithmetic played in expressing ILP systems, the quantifier-free fragment of the first-order theory of the real numbers with the

integers as a distinguished subset and with addition and order $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$ suffices to formulate MILP questions. There is a rich tradition of studying this theory. In 1999, Weispfenning [100] gave a quantifier elimination for first-order statements of $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$. This is related to Robinson and Zakon's complete first-ordered axiomatisations [87] of ordered abelian groups.

Boigelot, Bronne, and Rassart [15] and Boigelot and Wolper [18] gave an automata theoretic decision procedure for the first-order theory of $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$. A real number can be expressed in base 2 as a finite word over $\{0, 1\}$ followed by a separator symbol $(\star)$ and then an infinite expansion of the fractional part. As in the Presburger arithmetic case, we present an algorithm using base 2 expansions which generalizes easily to base r representations of real numbers. Thus, real vectors with n components are encoded as $\mathbf{w} = \mathbf{w_I} * \mathbf{w_F}$ where $\mathbf{w_I} \in (\{0, 1\}^n)^*$ and $\mathbf{w_F} \in (\{0, 1\}^n)^\omega$. We require that the lengths of the integer part of the encoding of all components of the vector be the same. Negative numbers are represented with 2's complement notation. If $\alpha \in \{0, 1, \star\}^\omega$ is a valid encoding of a real number, we write $[\alpha]_2$ for the value encoded by α. There are infinitely many such encodings for each vector of real numbers: the leading bit (sign bit) may be repeated arbitrarily many times; and, any fraction which has a finite representation has two encodings, one with infinitely many zeroes as its tail and one with infinitely many ones as its tail.

The basic objects, therefore, are infinite strings. Thus, we use Büchi automata to work on these strings (recall the definitions from Section 1.2). We restrict to a particular class of Büchi automata, defined in [15], which are especially well-suited for working on encodings of real numbers.

Definition 4.6. A **Real Vector Automaton** (RVA) for vectors in $\mathbb{R}^n$ is a Büchi

automaton over $\Sigma = (\{0, 1, \star\})^n$ such that every word $\mathbf{w}$ accepted by the automaton is of the form $\mathbf{w_I} \star \mathbf{w_F}$ where $\mathbf{w_I} \in (\{0, 1\}^n)^*$ and $\mathbf{w_F} \in (\{0, 1\}^n)^\omega$, and for every vector $\mathbf{x} \in \mathbb{R}^n$, either all encodings of $\mathbf{x}$ are accepted, or none are.

Slight modifications of the algorithms for Büchi automata show that given two RVA, we can construct a RVA corresponding to either the union or the intersection of their languages. Likewise, RVA recognisable languages are closed under complementation. As an example, Figure 4.1 depicts a RVA which accepts the language $\{w : w \text{ is an encoding for } 0.5\}$.

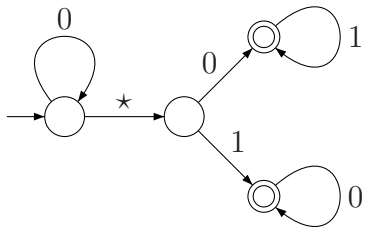

Figure 4.1: A RVA representing $\{\frac{1}{2}\}$.

Recall that in this section we are looking at connections between automata theoretic techniques, MILP, and the first-order theory of $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$. Since we have an automata representation of real numbers, it is prudent to check if the integers are a recognisable subset in this presentation. It is, in fact, easy to see that this is the case. Figure 4.2 gives the RVA whose language is the set of all encodings of integers.

Boigelot, Bronne, and Rassart [15] and Boigelot and Wolper [18] adapted the strategy from Section 4.1 to define an automata theoretic decision procedure for the first-order theory of $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$. Given a first-order formula in $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$, we construct a RVA representing the set of all vectors in $\mathbb{R}^n$ satisfying the formula. We proceed inductively, at each step building the RVA in two parts: one

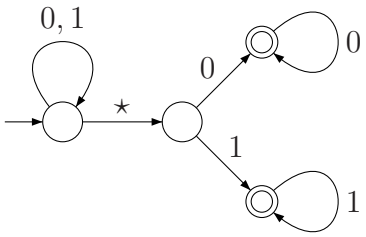

Figure 4.2: A RVA representing $\mathbb{Z}$.

accepting the integer parts of solutions, and one accepting the fractional parts.

Base case, Part 1: Automata for Equations. Suppose $\varphi \equiv \mathbf{a} \cdot \mathbf{x} = c$ for $\mathbf{a}, c$ as before. We write $\mathbf{x} = \mathbf{x_I} + \mathbf{x_F}$, where $\mathbf{x_I} \in \mathbb{Z}^n$ and $\mathbf{x_F} \in [0,1)^n$. For any encoding $\mathbf{w}$ of $\mathbf{x}$, $\mathbf{w} = \mathbf{w_I} \star \mathbf{w_F}$ and $\mathbf{w_I} \star \mathbf{0}^\omega$ is an encoding for $\mathbf{x_I}$, $\mathbf{0} \star \mathbf{w_F}$ is an encoding for $\mathbf{x_F}$. Define $m = \sum_{a_i<0} a_i$, $M = \sum_{a_i>0} a_i$. As each component of the vector $\mathbf{x_F}$ is in the interval $[0,1)$, and $\mathbf{a} \cdot \mathbf{x_I} + \mathbf{a} \cdot \mathbf{x_F} = c$, we have

$$m \leq \mathbf{a} \cdot \mathbf{x_F} \leq M \qquad \text{and} \qquad c - M \leq \mathbf{a} \cdot \mathbf{x_I} \leq c - m$$

Since all components of $\mathbf{x_I}$ are integers, $\gcd(a_1, \ldots, a_n) | \mathbf{a} \cdot \mathbf{x_I}$. Thus, if L is the set of all encodings of vectors $\mathbf{x}$ satisfying φ,

$$L = \bigcup_{d:\chi(d) \text{ holds}} (\{\mathbf{w_I} : \mathbf{a} \cdot [\mathbf{w_I} \star \mathbf{0}^\omega]_2 = d\} \cdot \{\star^n\} \cdot \{\mathbf{w_F} : \mathbf{a} \cdot [\mathbf{0} \star \mathbf{w_F}]_2 = c - d\})$$

where $\mathbf{w_I} \in (\{0,1\}^n)^*$, $\mathbf{w_F} \in (\{0,1\}^n)^\omega$, and

$$\chi(d) \iff ([c - M \leq d \leq c - m] \wedge (\exists i \in \mathbb{Z})(d = i \cdot \gcd(a_1, \ldots, a_n))).$$

Thus, the RVA representing the set of vectors satisfying φ can be decomposed into at most $m + M$ RVA, each of which consists of

1. A finite automaton over $\Sigma = \{0,1\}^n$ accepting all $\mathbf{w_I} \in \Sigma^*$ such that $\mathbf{a} \cdot [\mathbf{w_I} \star \mathbf{0}^\omega]_2$ is a solution to some linear equation,

2. a transition between the finite automaton from (1) and the Büchi automaton from (3) when read $\star^n$, and

3. a Büchi automaton over $\Sigma = \{0,1\}^n$ accepting all $\mathbf{w_F} \in \Sigma^\omega$ such that $\mathbf{a} \cdot [\mathbf{0} \star \mathbf{w_F}]_2$ is a solution to some linear equation.

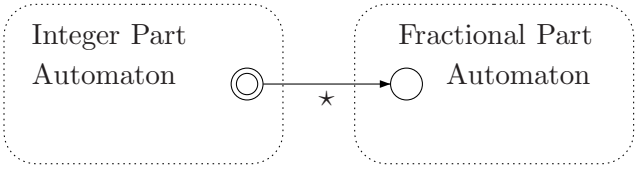

Figure 4.3: The decomposition of RVA.

The techniques from Section 4.1 may be used to construct the integer part automaton. We now define Büchi automata recognising the fractional solutions $\mathbf{x} \in [0,1)^n$ to $\mathbf{a} \cdot \mathbf{x} = c - d$ for a given d. For convenience, we write $k = c - d$. Each state in the automaton will correspond to an integer s such that any vector $\mathbf{x} \in [0,1)^n$ encoded by a path starting at that state and continuing infinitely long satisfies $\mathbf{a} \cdot \mathbf{x} = s$. The states in the automata will be labelled by integers in $[m, M]$ and all the states are accepting states; the initial state is $\iota = k$. For $s \in [m, M]$ and $\mathbf{b} \in \{0,1\}^n$ we define $\delta(s, \mathbf{b}) = 2s - \mathbf{a} \cdot \mathbf{b}$ if $m \leq 2s - \mathbf{a} \cdot \mathbf{b} \leq M$, and $\delta(s, \mathbf{b}) = \emptyset$ otherwise. We call the above Büchi automaton $\mathcal{A}_\varphi^{F,k}$ for $\varphi \equiv \mathbf{a} \cdot \mathbf{x} = c$ and a given value of k. Note that any infinite run on $\mathcal{A}_\varphi^{F,k}$ is an accepting run. The theorem below is suggested by the results in [15] and [18] but is not made explicit in either.

Theorem 4.7. *Given $\varphi \equiv \mathbf{a} \cdot \mathbf{x} = c$, d such that $\chi(d)$ holds, $k = c - d$, and $\alpha \in (\{0,1\}^n)^\omega$, then $\alpha \in L(\mathcal{A}_\varphi^{F,k})$ if and only if $\mathbf{a} \cdot [\alpha]_2 = k$.*

Proof. For the forward direction, suppose that $\mathbf{x} \in [0,1)^n$ is such that $\mathbf{a} \cdot \mathbf{x} = k$ and $[\mathbf{w}]_2 = \mathbf{x}$. To show there is an infinite run of $\mathcal{A}_\varphi^{F,k}$ on $\mathbf{w}$ it suffices to prove that

at each state s of the run, on input $\mathbf{w_i}$, $2s - \mathbf{a} \cdot \mathbf{w_i} \in [m, M]$. We go by induction and prove the additional statement that if we transition to s' from s and if $\mathbf{y}, \mathbf{z}$ are encoded by paths labelled by bits in $\mathbf{w}$ starting from s, s' (respectively), then if $\mathbf{a} \cdot \mathbf{y} = s$ it must be that $\mathbf{a} \cdot \mathbf{z} = s'$. For the base case, since $\iota = k$ and $\mathbf{w_0}$ is the vector of most significant bits of an encoding of a solution to $\mathbf{a} \cdot \mathbf{x} = k$, we have

$$\frac{1}{2}\mathbf{a} \cdot [\mathbf{w_0}]_2 + \mathbf{a} \cdot [\mathbf{0} \star \mathbf{0}\mathbf{w}_{tail}]_2 = k.$$

By definition of m and M and since $[\mathbf{w}_{tail}]_2$ can be at most $\frac{1}{2}$,

$$\frac{m}{2} \leq \mathbf{a} \cdot [\mathbf{0} \star \mathbf{0}\mathbf{w}_{tail}]_2 \leq \frac{M}{2}.$$

Therefore, $m \leq 2k - \mathbf{a} \cdot [\mathbf{w_0}]_2 \leq M$ and there is a transition out of ι on the initial input of $\mathbf{w}$. We now prove the second statement required in the base case. Let $s_1 = \delta(k, \mathbf{w_0})$. Any path from s_1 is labelled by a word $\mathbf{u}$ such that if $\mathbf{v}$ labels a path starting at k and going to s_1, $\mathbf{v} = \mathbf{w_0}\mathbf{u}$. If $\mathbf{v}, \mathbf{u}$ are fractional binary representations of $\mathbf{y}, \mathbf{z}$ respectively, $\mathbf{y} = \frac{1}{2}(\mathbf{w_0} + \mathbf{z})$. Hence, if $\mathbf{a} \cdot \mathbf{y} = k$,

$$\mathbf{a} \cdot \mathbf{z} = 2\mathbf{a} \cdot \mathbf{y} - \mathbf{a} \cdot \mathbf{w_0} = 2k - \mathbf{a} \cdot \mathbf{w_0} = s_1.$$

Thus, if $\mathbf{y}$ satisfies $\mathbf{a} \cdot \mathbf{y} = k$ and is encoded by a path starting at state k and if $\mathbf{z}$ is encoded by the tail of the path starting at s_1 (truncating the first bit), then $\mathbf{z}$ satisfies $\mathbf{a} \cdot \mathbf{z} = s_1$.

For the inductive step, we suppose that the current state of $\mathcal{A}_\varphi^{F,k}$ is some s_i and the current input bit is $\mathbf{w_i}$. By the inductive hypothesis, at each previous state there was a transition out of the state on the input bit. Hence we have a path through states $k, s_1, s_2, \ldots, s_i$ labelled by the first i bits of $\mathbf{w}$. Moreover, there is a relationship between words starting at each previous state and the satisfiability of a linear equation whose left hand side is $\mathbf{a} \cdot \mathbf{x}$. The current input is the $(i+1)^{st}$ bit, $\mathbf{w_i}$, of an encoding of a solution to $\mathbf{a} \cdot \mathbf{x} = k$. By the inductive hypothesis, it

is also the most significant bit of an encoding of a solution to the linear equation $\mathbf{a} \cdot \mathbf{x} = s_i$. As before, $m \leq 2s_i - \mathbf{a} \cdot \mathbf{w_i} \leq M$ and hence δ gives a transition out of s_i. Let $s_{i+1} = \delta(s_i, \mathbf{w_i})$. The second part of the inductive claim is proved similarly. Hence, the induction holds and we have shown that if $\mathbf{w}$ is an encoding of a solution to $\mathbf{a} \cdot \mathbf{x} = k$ then there is a run of $\mathcal{A}_\varphi^{F,k}$ on $\mathbf{w}$ which is successful.

Conversely, suppose $\rho = k, s_1, s_2, \ldots$ is a successful run of $\mathcal{A}_\varphi^{F,k}$ on given input $\mathbf{w} \in (\{0,1\}^n)^\omega$. We will prove that $\mathbf{w}$ encodes a solution to $\mathbf{a} \cdot \mathbf{x} = k$. Since ρ is successful, there is some accepting state which appears infinitely often in ρ.

Lemma 4.8. *Suppose the states s and s' are connected by a path $\mathbf{b_0}, \ldots, \mathbf{b_j}$. If $\alpha, \alpha' \in (\{0,1\}^n)^\omega$ label paths starting at s, s' (respectively) and $\alpha = \mathbf{b_0} \cdots \mathbf{b_j} \alpha'$ then*

$$\mathbf{a} \cdot [\alpha]_2 - s = \frac{\mathbf{a} \cdot [\alpha']_2 - s'}{2^{j+1}}.$$

Proof. We proceed by induction on j. For $j = 0$, $[\alpha]_2 = \frac{1}{2}([\alpha']_2 + \mathbf{b_0})$. Moreover, by the transition relation, $s' = 2s - \mathbf{a} \cdot \mathbf{b_0}$. Hence, $\mathbf{a} \cdot [\alpha]_2 - s = \frac{1}{2}(\mathbf{a} \cdot [\alpha']_2 - s')$. For the inductive step, we have that states s and s' are connected by a path labelled by $\mathbf{b_0}, \ldots, \mathbf{b_{j+1}}$ and that α, α' label paths starting at s, s' (respectively) such that $\alpha = \mathbf{b_0}, \ldots, \mathbf{b_j}, \alpha'$. Let s'' be such that $\delta(s'', \mathbf{b_j}) = s'$ and let $\alpha'' = \mathbf{b_j} \cdot \alpha'$. The base case of the induction gives that $\mathbf{a} \cdot [\alpha'']_2 - s'' = \frac{1}{2}(\mathbf{a} \cdot [\alpha']_2 - s')$. Moreover, the states s and s'' are connected by a path of length j, hence the inductive hypothesis implies that $\mathbf{a} \cdot [\alpha]_2 - s = \frac{1}{2^j}(\mathbf{a} \cdot [\alpha'']_2 - s'')$. Putting these together, we get the desired result. □

Let s' be a state appearing infinitely often in ρ. There are are infinitely many indices ℓ_j for which $s' = s_{\ell_j}$. We write ρ_{ℓ_j} for the sequence of states in the ρ beginning with s_{ℓ_j} (the tail of the run) and we write α_{ℓ_j} for the corresponding

sequence of transition labels. By Lemma 4.8, for each ℓ_j,

$$\mathbf{a} \cdot [\alpha]_2 - k = \frac{\mathbf{a} \cdot [\alpha_{\ell_\mathbf{j}}]_2 - s'}{2^{\ell_j+1}}.$$

But, $|\mathbf{a} \cdot [\alpha_{\ell_\mathbf{j}}]_2 - s'| \leq |\mathbf{a} \cdot [\alpha_{\ell_\mathbf{j}}]_2| + |s'| \leq 2M$, and therefore $\mathbf{a} \cdot [\alpha]_2 - k \leq \frac{M}{2^{\ell_j}}$ for all j. Hence, $\mathbf{a} \cdot [\alpha]_2 - k = 0$, as required. □

We now summarize the construction of RVA for equations. Given $\varphi \equiv \mathbf{a} \cdot \mathbf{x} = c$, we compute $gcd(a_1, \ldots, a_n)$ and $m = \sum_{a_i<0} a_i$, $M = \sum_{a_i>0} a_i$. For each d a multiple of $gcd(a_1, \ldots, a_n)$ satisfying $c - M \leq d \leq c - m$ we build a RVA as follows. The integer part of the automaton is a finite automaton with a single accepting state which accepts all encodings of integer vectors satisfying $\mathbf{a} \cdot \mathbf{x} = d$. The fractional part of the automaton is $\mathcal{A}_\varphi^{F,c-d}$ from above. There is a transition labelled by $\star^n$ from the accepting state of the integer part of the automaton to the initial state of the fractional part of the automaton. The resulting automaton is a RVA whose language is a subset of all encodings of real numbers satisfying $\mathbf{a} \cdot \mathbf{x} = c$. Once RVA are built for each d, the union automaton is taken and is $\mathcal{A}_\varphi$.

A few optimizations may be made in the construction of $\mathcal{A}_\varphi$. First, notice that the states and transitions can be shared among the integer part automata for various values of d, and likewise among the fractional part automata for various values of d. Figure 4.4 demonstrates this sharing of states. Also, computing the greatest common divisor may be costly time-wise. We may relax the condition on the value of d and only require it to be between $c - M$ and $c - m$. This would result in more transitions between the integer and fractional parts of $\mathcal{A}_\varphi$. The benefit of such an optimization would depend on the relative sizes of the coefficients $a_1, \ldots, a_n, c$.

We include in Figure 4.5 a fully worked example of a RVA representing the

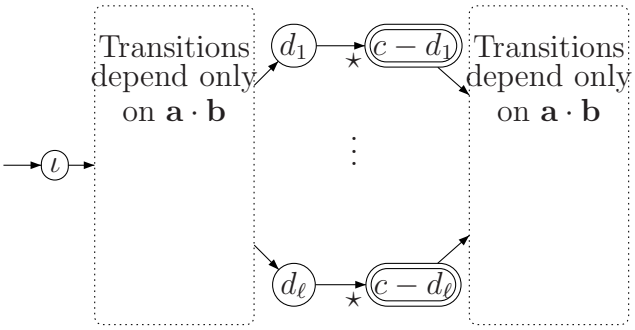

Figure 4.4: Sharing states in $\mathcal{A}_\varphi$.

equation $x + y = 3$. Note the decomposition into integer and fractional parts, and the reuse of states and transitions.

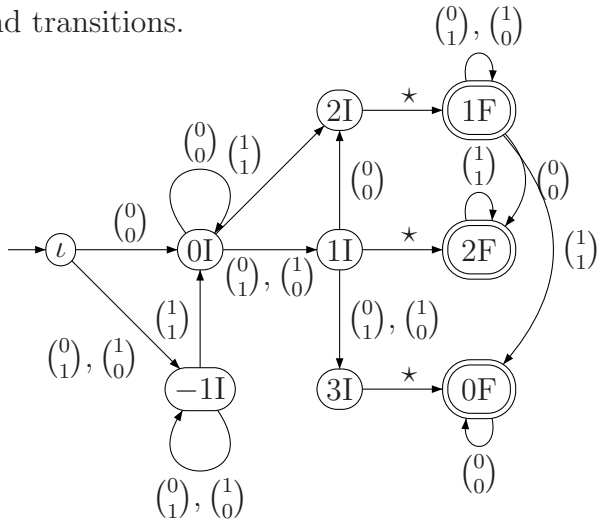

Figure 4.5: A RVA representing the equation $x + y = 3$.

Base case, Part 2: Automata for Inequalities. The construction for inequalities is very similar to what we saw in Part 1 above. Let $\varphi \equiv a_1x_1 + \cdots + a_nx_n \leq c$, which we abbreviate as $\mathbf{a} \cdot \mathbf{x} \leq c$. As before, we partition the set of all encodings of solutions:

$$\bigcup_{d:\chi(d) \text{ holds}} (\{\mathbf{w_I} : \mathbf{a} \cdot [\mathbf{w_I} \star \mathbf{0}^\omega]_2 \leq d\} \cdot \{\star^n\} \cdot \{\mathbf{w_F} : \mathbf{a} \cdot [\mathbf{0} \star \mathbf{w_F}]_2 \leq c - d\})$$

where $\mathbf{w_I} \in (\{0,1\}^n)^*$, $\mathbf{w_F} \in (\{0,1\}^n)^\omega$, and $\chi(d) \iff (c - M \leq d \leq c - m)$. Again, we can construct the RVA for φ by concatenating finite automata accepting encodings of integer solutions to some linear inequality with Büchi automata representing sets of vectors in $[0,1]^n$ which satisfy some linear inequality. Since finite automata for inequalities were discussed in Section 4.1, it suffices to modify the construction from Part 1 of the base case to obtain Büchi automata accepting the fractional solutions of inequalities. Given k, we define $\mathcal{A}_\varphi^{F,k} = (\{m, m+1, \ldots, M\}, k, \delta, \{m, m+1, \ldots, M\})$, where for each $s \in \{m, \ldots, M\}, \mathbf{b} \in \{0,1\}^n$, $\delta(s, \mathbf{b}) = 2s - \mathbf{a} \cdot \mathbf{b}$ if $m \leq 2s - \mathbf{a} \cdot \mathbf{b}$ (and empty otherwise).

We give evidence for the correctness of this definition; a full proof of correctness parallels that in Part 1 of the base case. Suppose $s, s' \in \{m, \ldots, M\}$ are connected by a transition on input $\mathbf{b}$ and let α, α' be words labelling paths starting at s, s' (respectively) such that $\alpha = \mathbf{b}\alpha'$. Hence $[\alpha]_2 = \frac{1}{2}(\mathbf{b} + [\alpha']_2)$. If $\mathbf{a} \cdot [\alpha]_2 \leq s$,

$$s \geq \mathbf{a} \cdot [\alpha]_2 = \frac{1}{2}(\mathbf{b} + [\alpha']_2) = \frac{1}{2}(2s - s' + [\alpha']_2)$$

and $\mathbf{a} \cdot [\alpha']_2 \leq s'$ This supports the intuition that for each $s \in \{m, \ldots, M\}$, the vectors encoded by infinite paths starting from s satisfy $\mathbf{a} \cdot \mathbf{x} \leq s$. For $s' > \alpha'$, note that since $\mathbf{x} \in [0,1]^n$, the set of solutions to $\mathbf{a} \cdot \mathbf{x} \leq s'$ is the same as the set of solutions to $\mathbf{a} \cdot \mathbf{x} \leq \alpha'$.

Before we generalize the automata constructions above to arbitrary first-order formulas of $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$, let us examine the structure of the RVA built so far. A notion from Büchi automata theory [93], [94] will be relevant.

Definition 4.9 (Staiger, Wagner; 1974. Staiger; 1983.)**.** A Büchi automaton is **weak** if there is a partition of its states S into disjoint subsets $Q_1, \ldots, Q_m$ such that for each Q_i either $Q_i \subset F$ or $Q_i \cap F = \emptyset$; and such that there is a partial order on $\{Q_1, \ldots, Q_m\}$ with $Q_j \leq Q_i$ if Q_j is reachable from Q_i.

Theorem 4.10 (Boigelot, Jodogne, Wolper; 2001)**.** *The RVA constructed above for equations and inequalities with real-valued free variables are weak Büchi automata.*

Proof. Suppose the RVA $\mathcal{A} = (S, \iota, \delta, F)$ has been constructed as above. Partition the set of states S into strongly connected components. Note that each strongly connected component will be either entirely within the integer part of the RVA or entirely within the fractional part of the RVA. By definition, all nodes in the integer part are non-accepting and all nodes in the fractional part are accepting. Hence, the first requirement of Definition 4.9 is met. To fulfill the second condition, consider the reachability relation on the strongly connected components. By definition of strongly connected, this relation is reflexive, anti-symmetric, and transitive. Hence, it is a partial order and respects reachability. □

Weak Büchi automata have properties which can be used to control the size of RVA built for general first-order formulas. First, we note that the simple product construction for union and intersection preserves weakness of Büchi automata. Moreover, to complement a deterministic weak Büchi automata, it is sufficient to switch all accepting and non-accepting states (as in the deterministic finite automata case, but unlike deterministic Büchi automata). This complementation operation has no cost in terms of size of the automaton. Recall that in general, we cannot determinize Büchi automata. In order to apply the efficient complementation, we need to obtain deterministic weak Büchi automata. The following definition and theorems tell us that we will be able to do just this.

Definition 4.11. A **co-Büchi automaton** is defined by (S, ι, δ, F) similarly to a Büchi automaton, except that a computation on an infinite word α is successful it it infinitely often *avoids* the set of accepting states.

It is not hard to see that, given a weak Büchi automaton, switching the accepting states with non-accepting states yields a co-Büchi automaton which accepts exactly the same infinite words. Even though general Büchi automata cannot be determinized, the following theorem from [74] and [68] shows that each weak Büchi automaton has an equivalent deterministic Büchi automaton.

Theorem 4.12 (Miyano, Hayashi; 1984. Kupferman, Vardi; 1997)**.** *Weak Büchi automata can be determinized by a "breakpoint" construction.*

Proof. We fix a finite alphabet Σ. Let $\mathcal{A} = (S, \iota, \delta, F)$ be a weak Büchi automaton over Σ, so $\mathcal{A}' = (S, \iota, \delta, S \setminus F)$ is the equivalent weak co-Büchi automaton. We define the deterministic Büchi automaton $\mathcal{A}'' = (S'', \iota'', \delta'', F'')$ as follows. Put $S'' = 2^S \times 2^S$, $\iota'' = (\{\iota\}, \emptyset)$, $F'' = 2^S \times \{\emptyset\}$. To define the transition function we use the following auxiliary sets which may be associated to any subsets Q, R of S:

$$T_Q = \{p \in S : \exists q \in Q\,(p \in \delta(q, a))\} \quad \text{and} \quad U_{Q,R} = \{p \in S : \exists r \in R\,(p \in \delta(r, a))\}.$$

For $(Q, \emptyset) \in S''$ and $a \in \Sigma$, let $\delta''((Q, \emptyset), a) = (T_Q, T_Q \cap F)$. For $(Q, R) \in S''$ with $R \neq \emptyset$ and $a \in \Sigma$, put $\delta''((Q, R), a) = (T_Q, U_{Q,R} \cap F)$. A breakpoint of a run of $\mathcal{A}''$ occurs when the run enters a state of the form $(Q, \emptyset)$. Intuitively, if $\mathcal{A}''$ is in state (Q, R) then R is the set of states of $\mathcal{A}$ reachable in $\mathcal{A}'$ by a run whose corresponding run in $\mathcal{A}''$ hasn't passed through a state in F' since the last breakpoint. It is easy to see that $L(\mathcal{A}'') = L(\mathcal{A}') = L(\mathcal{A})$. □

While Theorem 4.12 does not produce a *weak* deterministic automaton, if we apply it to a RVA we get something almost as good.

Definition 4.13. A Büchi automaton is called **inherently weak** if its underlying directed graph has no reachable strongly connected components which include both accepting and non-accepting states.

A deterministic inherently weak Büchi automaton, is easily transformed into a deterministic weak Büchi automaton: for each state, if it is in a strongly connected component with at least one accepting state, add it to the set of accepting states. The following theorem from [16] uses a topological characterization of sets of infinite words recognisable by inherently weak Büchi automata.

Theorem 4.14 (Boigelot, Jodogne, Wolper; 2001)**.** *Every deterministic RVA representing a set first-order definable in* $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$ *is inherently weak.*

Thus far, we have produced deterministic RVA representing sets of solutions to linear equations and inequalities in $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$. These RVA are inherently weak by Theorem 4.14 and since they are deterministic, they can be assumed to be weak Büchi automata. Therefore, the union, intersection, and complementation constructions are efficient and we have RVA for sets definable in the quantifier-free fragment of the first-order logic of $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$. Note that to ensure that the complement language of a RVA accepts only strings which correspond to encodings of real vectors, we intersect the complement automaton with the RVA for $\mathbb{R}^n$. To build automata for existential formulas, we use the projection operation. The resulting automaton is not necessarily deterministic but is still weak. As in Section 4.1, we add transitions from the initial state to some of its reachable states to ensure that this weak Büchi automaton is a RVA. The resulting weak RVA accepts exactly all encodings of solutions to the existential formula. Thus, all first-order formulas of $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$ have corresponding RVA. As in the case of Presburger arithmetic, the translation of formulas into automata solves the satisfiability problem. In particular, checking for the existence of a lasso in the corresponding RVA gives a witness corresponding to a rational solution if a formula is indeed satisfiable. Several automata implementations are available for deciding the first-order theory of $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$: LASH [17], LIRA [7], and MONA [66] are popular examples.

4.3 Automata and the p-adics

The **p-adic** numbers are defined as completions of the rational numbers with respect to the p-adic norms. They were introduced by Hensel in 1897 [46] and explored further in his book of 1908 [47]. The study of the p-adic numbers was originally motivated by analogy to power series, but has since become central in algebraic number theory. For a fixed prime p, the p-adic valuation $ord_p : \mathbb{Q} \to \mathbb{Z}$ is defined as follows: for $m, n \in \mathbb{Z}^{\neq 0}$, $ord_p(0) = \infty$, $ord_p(m) = \max\{r : p^r | m\}$, $ord_p(\frac{n}{m}) = ord_p(n) - ord_p(m)$. The corresponding p-adic norm for $x \in \mathbb{Q}$ is given by $|0|_p = 0$ and $|x|_p = p^{-ord_p x}$ (for $x \neq 0$). Completing $\mathbb{Q}$ with respect to $|\cdot|_p$ yields the field of p-adic numbers, $\mathbb{Q}_p$. The ring of p-adic integers, $\mathbb{Z}_p$, consists of those p-adic numbers with norm less than or equal to 1: $\mathbb{Z}_p = \{x \in \mathbb{Q}_p : |x|_p \leq 1\}$. In the following, p-adic expansions will come into play (in the same way that base 2 expansions were key in Sections 4.1 and 4.2). Every p-adic number has a unique p-adic expansion

$$x = x_{-r}p^{-r} + x_{-r+1}p^{-r+1} + \cdots + x_{-1}p^{-1} + x_0 + x_1 p + x_2 p^2 + \cdots$$

with x_i a natural number between 0 and $p-1$ for all i, and $x_{-r} \neq 0$ where $r = |x|_p$. Moreover, $x \in \mathbb{Z}_p$ if and only if its p-adic expansion contains no negative powers of p. A good introduction to the p-adic numbers is the textbook [42].

It is natural to wonder about the first-order theory of p-adic fields, as we did about Presburger arithmetic and other structures in previous sections. In [4], Ax and Kochen give a complete and computable axiomatisation of the first-order theory of the valued field $\mathbb{Q}_p$ with the valuation group $\mathbb{Z}_p$. Moreover, they show in [3], [4] that the first-order theory of $\mathbb{Q}_p$ is decidable. A similar result was proved independently by Ershov in [39]. This early work used abstract tools of model theory, including ultraproducts and model-completeness, which imply decidability but

are not constructive. Cohen's 1969 paper [30] uses primitive recursive quantifier elimination to explicitly give a decision procedure for the first-order theory of $\mathbb{Q}_p$. Weispfenning [97], [98] simplified Cohen's proof and weakened some hypotheses. Moreover, he explored the complexity of the quantifier elimination [99]: there are double exponential time and double exponential space procedures for elimination of quantifiers in linear formulas; and these are the best bounds possible. In that paper, Weispfenning also proves that the full quantifier elimination for the theory of $\mathbb{Q}_p$ takes double exponential space. In [36], Dubhashi gives similar complexity results for a different fragment of the theory of $\mathbb{Q}_p$, one that can be seen as analogous to Presburger arithmetic. Also, he shows that the full quantifier elimination for the first-order theory of $\mathbb{Q}_p$ (augmented with an additional relation) takes double exponential time and double exponential space.

In this section, we give an alternate proof of the decidability of the first-order theory of $\mathbb{Q}_p$ using automata techniques. As in Section 4.1, we restrict our attention to $\mathbb{Q}_p$ with addition and the distinguished ring of integers (analogous to addition and order), but not including multiplication. We provide translation schemes from the p-adics to automata. Then, the techniques discussed in the previous section yield decision procedure for the first-order theories of the corresponding structure. We note that, as presented, these decision procedures have nonelementary time and space complexity. However, it is possible that an analysis analogous to that of Klaedtke's in the Presburger arithmetic case [65] may bring down this complexity.

Theorem 4.15. *There is an algorithm which, given any first-order formula of* $(\mathbb{Q}_p; \mathbb{Z}_p, +, 0, 1)$*, produces an automaton whose language represents exactly those p-adic numbers satisfying the formula.*

Corollary 4.16. *The first-order theory of* $(\mathbb{Q}_p; \mathbb{Z}_p, +, 0, 1)$ *is decidable.*

To prove Theorem 4.15, we need to specify how we will encode p-adic numbers. Given $x \in \mathbb{Q}_p$ with p-adic expansion $x_{-r}p^{-r} + x_{-r+1}p^{-r+1} + \cdots + x_{-1}p^{-1} + x_0 + x_1 p + x_2 p^2 + \cdots$, we represent it by the infinite string $x_{-r} \cdots x_{-1} \star x_0 x_1 \cdots$. Note that each p-adic number is represented by an infinite string over $\{0, 1, \ldots, p-1, \star\}$. Each p-adic number has infinitely many representations corresponding to a padding of 0s as coefficients of p^{-t} for large t. However, unlike the case of real numbers, each p-adic numbers has a unique "fractional" representation. As before, when we encode vectors of p-adic numbers we require that they have a common least power of p; that is, the $\star$ symbol appears at the same position in all components. Since we use infinite strings to represent p-adic numbers, the automata translation needs to use automata on infinite strings. To make the translation smoother, we recall a variant of Büchi automata.

Definition 4.17. A **Müller automaton** over a finite alphabet Σ is given by $\mathcal{M} = (S, \iota, \Delta, \mathcal{F})$ where S is a finite set of states, ι is the designated initial state, $\Delta : S \times \Sigma \to S$ is the transition function, and $\mathcal{F} \subset \mathcal{P}(S)$ is the set of accepting sets of states. Inputs to $\mathcal{M}$ are infinite words $\alpha \in \Sigma^\omega$. The **computation** of $\mathcal{M}$ on input α is an infinite sequence of states $s_0, s_1, s_2 \ldots$ such that $s_0 = \iota$ and for each i, $(s_i, \sigma_{\mathbf{i}}, s_{i+1}) \in \Delta$. A computation of $\mathcal{M}$ is successful if the set of states that it enters infinitely many times is a member of $\mathcal{F}$; α is **accepted** by $\mathcal{M}$ if the computation of $\mathcal{M}$ on α is successful.

Implicit in the definition is that any Müller automaton is deterministic. Moreover, we note that any deterministic Büchi automaton can be transformed into an equivalent Müller automaton by setting $\mathcal{F} = \{X \subset S : X \cap F \neq \emptyset\}$. McNaughton's fundamental theorem [73] says that any Büchi recognisable language is Müller recognisable and vice versa. For more on properties of automata on infinite words, see [57].

Our strategy will be to translate the atomic formulas of $(\mathbb{Q}_p; \mathbb{Z}_p, +, 0, 1)$ to equivalent Müller automata. By closure properties of Büchi (and hence Müller) recognisable languages, any first-order formula will then be associated with an automaton. Notice that the set $\mathbb{Z}_p$ is recognised by the automaton $(\{\iota, f\}, \iota, \delta, \{f\})$ where $\delta(\iota, 0) = \iota$, $\delta(\iota, \star) = f$, and for $b \in \{0, \ldots, p-1\}$, $\delta(f, b) = f$. Since we have a representation for the domain and for $\mathbb{Z}_p$, it remains to specify the Müller automaton recognising the graph of addition. For brevity, we will give the automata for $x + y = 0$ where $p = 2$. It is not hard (but notationally cumbersome) to generalize to several variables. The transition relation for the automaton for $x + y = 0$ is represented in Figure 4.6; the accepting sets of states are $\mathcal{F} = \{\{0\}, \{1\}, \{0, 1\}\}$.

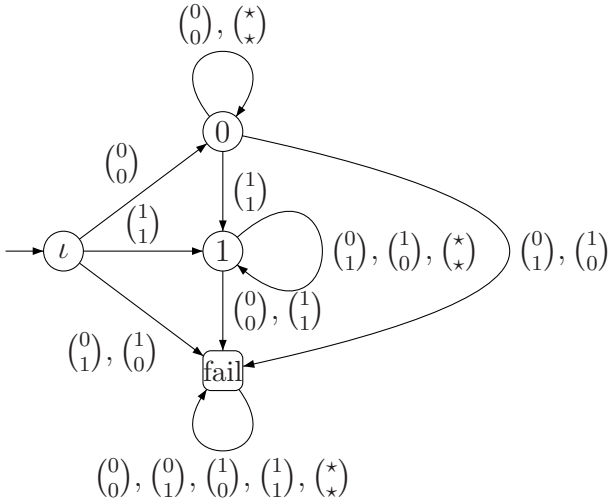

Figure 4.6: A Müller automaton representing p-adic solutions to $x + y = 0$.

For general p, more states must be added to represent a possible carry value of $0, \ldots, p-1$. In this case, transitions will be based on the sum of the input bits and carry, $\mod p$. The translation of first-order formulas of $(\mathbb{Q}_p; \mathbb{Z}_p, +, 0, 1)$ to automata is now complete.

4.4 Automata and formal power series

The connection between the p-adic numbers and formal power series leads naturally to another arena where automata techniques may be used. A **formal power series** over a field F is an infinite sum $\sum_i a_i x^i$ where $a_i \in F$ and where convergence considerations are ignored. Thus, a formal power series can be thought of as an infinite sequence of coefficients $\{a_i\}$. To strengthen the analogy with p-adic numbers, we consider **formal Laurent series**, in which finitely many negative powers of x are allowed: $\sum_{i>-r} a_i x^i$. In this case, we can represent the series as an infinite sequence with a distinguished symbol $\star$ to distinguish coefficients of negative powers from those of positive powers $a_{-r}, a_{-r+1}, \ldots, a_{-1}, \star, a_0, a_1, \ldots$. We use the notation from [37] and denote by $F[[x]]$ the set of all formal power series over F and by $F((x))$ the set of all formal Laurent series over F. In particular, consider the case where F is the finite field $\mathbb{F}_p$ for some prime p. A valuation may be associated with the set of formal Laurent series, $ord : \mathbb{F}_p((x)) \to \mathbb{Z}$ where $ord\left(\sum_{i>-r} a_i x^i\right) = \min\{n : a_n \neq 0\}$. The "ring of integers" then becomes $\mathbb{F}_p[[x]]$. Addition and multiplication may be defined on the set of formal Laurent series by treating them as generalizations of polynomials: addition is performed term-by-term, and the multiplication operation is a more complicated version of polynomial multiplication.

Theorem 4.18. *The first-order theory of* $(\mathbb{F}_p((x)); \mathbb{F}_p[[x]], +, 0)$ *is decidable. In particular, there is an algorithm which produces from a first-order formula in this language a Büchi automaton encoding all tuples satisfying the formula in* $(\mathbb{F}_p((x)); \mathbb{F}_p[[x]], +, 0)$.

Proof. As in the p-adic case, it suffices to provide a representation of elements of

$\mathbb{F}_p((x))$ in which $\mathbb{F}_p[[x]]$ is a recognisable subset and $+$ is a recognisable operation. We have already alluded to the solution: any formal Laurent series is represented as an infinite string over the alphabet $\{0, \ldots, p-1, \star\}$. It is easy to build an automaton recognising correctly formed strings and an automaton recognising strings representing formal power series (no nonzero coefficient may appear before $\star$). It remains to exhibit an automaton recognising the addition operation. Figure 4.7 presents a Büchi automaton recognising the graph of addition for $p = 2$.

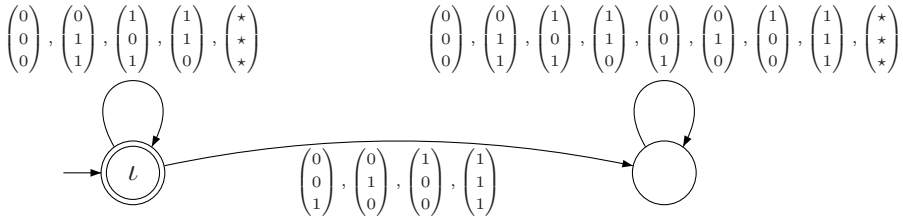

Figure 4.7: A Büchi automaton recognising the graph of addition for $p = 2$.

□

4.5 Conclusion

This chapter began by discussing the interactions of the first-order theories of $(\mathbb{Z}; +, \leq, 0, 1)$ and $(\mathbb{R}; \mathbb{Z}, +, \leq, 0, 1)$ with solutions to integer and mixed-integer linear programming problems. We saw that the logical formulation of linear programming problems allows extensions using first-order definitions, and that the associated automata can be used to solve these extensions as well. The translation paradigm was used to provide a new decision procedure for the first-order theory of $(\mathbb{Q}_p; \mathbb{Z}_p, +, 0, 1)$. We also adapted it for formal power series and formal Laurent series over a finite field and thus gave a decision procedure for the first-order theory of these objects under addition.

BIBLIOGRAPHY

[1] P.A. Abdulla, K. Čerāns, B. Jonsson, and Y. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.

[2] L. Aceto, A. Burgueno, and K.G. Larsen. Model checking via reachability testing for timed automata. In B. Steffen, editor, *Proceedings of 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, volume 1384 of *LNCS*, pages 263–280. Springer-Verlag, 1998.

[3] J. Ax and S. Kochen. Diophantine problems over local fields I. *American Journal of Mathematics*, 87(3):605–630, July 1965.

[4] J. Ax and S. Kochen. Diophantine problems over local fields II: A complete set of axioms for p-adic number theory. *American Journal of Mathematics*, 87(3):631–648, July 1965.

[5] V. Bárány. A hierarchy of automatic words having a decidable MSO theory. In D. Caucal, editor, *Online Proceedings of 11th Journées Montoises, Rennes*, 2006.

[6] V. Bárány. *Automatic Presentations of Infinite Structures*. Diploma thesis, DWTH Aachen, September 2007.

[7] B. Becker, C. Dax, J. Eisinger, and F. Klaedtke. LIRA: Linear integer/ real arithmetic solver. `http://lira.gforge.avacs.org/`, June 2007.

[8] C.H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, pages 525–532, 1973.

[9] S. Berezin, V. Ganesh, and D.L. Dill. Online proof-producing decision procedure for mixed-integer linear arithmetic. In H. Garavel and J. Hatcliff, editors, *Proceedings of 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *LNCS*, pages 521–537. Springer-Verlag, April 2003.

[10] A. Blaas and Y. Gurevich. Program termination and well partial orderings. *ACM Transactions on Computational Logic*, pages 1–25, December 2006.

[11] A. Blumensath. *Automatic Structures*. Diploma thesis, RWTH Aachen, October 1999.

[12] A. Blumensath. Prefix-recognisable graphs and monadic second-order logic. Technical Report AIB-2001-06, RWTH Aachen, May 2001.

[13] A. Blumensath and E. Grädel. Automatic structures. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science*, pages 51–62. IEEE Computer Society, 2000.

[14] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, 37:641–674, 2004.

[15] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems (extended abstract). In O. Grumberg, editor, *Proceedings of 9th International Conference on Computer Aided Verification*, volume 1254 of *LNCS*, pages 167–177. Springer-Verlag, June 1997.

[16] B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of 1st International Joint Conference on Automated Reasoning*, volume 2083 of *LNCS*, pages 611–625. Springer-Verlag, June 2001.

[17] B. Boigelot, L. Latour, and A. Legay. LASH: The Liège automata-based symbolic handler. `http://www.montefiore.ulg.ac.be/~boigelot/research/lash`.

[18] B. Boigelot and P. Wolper. Representing arithmetic constraints with finite automata: An overview. In P.J. Stuckey, editor, *Proceedings of 18th International Conference on Logic Programming*, volume 2401 of *LNCS*, pages 1–19. Springer-Verlag, July 2002.

[19] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In A.W. Mazurkiewicz and J. Winkowski, editors, *Proceedings of 8th International Conference on Concurrency Theory*, volume 1243 of *LNCS*, pages 135–150. Springer-Verlag, 1997.

[20] A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic, and finite automata. In H. Kirchner, editor, *Proceedings of 21st*

International Colloquium on Trees in Algebra and Programming, volume 1059 of *LNCS*, pages 30–43. Springer-Verlag, 1996.

[21] R. Brinkmann and R. Dreschler. RTL-datapath verification using integer linear programming. In *Proceedings of Design Automation Conference, 2002: 7th Asia and South Pacific and the 15th International Conference on VLSI Design*. IEEE Computer Society, January 2002.

[22] J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift Math. Logik und Grundlagen det Mathematik*, pages 66–92, 1960.

[23] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of International Congress in Logic, Methodology and Philosophy of Science, 1960*, pages 1–12, 1962.

[24] W. Calvert, S.S. Goncharov, and J.F. Knight. Computable structures of Scott rank ω_1^{CK} in familiar classes. In *Advances in Logic (Proceedings of North Texas Logic Conference, October 8-10, 2004)*, volume 425 of *Contemporary Mathematics*, pages 49–66. American Mathematical Society, 2007.

[25] D. Caucal. On infinite transition graphs having a decidable monadic theory. In F.M. auf der Heide and B. Monien, editors, *Proceedings of 23rd International Colloquium on Automata, Languages, and Programming*, volume 1099 of *LNCS*, pages 194–205. Springer-Verlag, July 1996.

[26] D. Caucal. On infinite graphs having a decidable monadic theory. In K. Diks and W. Rytter, editors, *Proceedings of 27th International Symposium on Mathematical Foundations of Computer Science*, volume 2420 of *LNCS*, pages 165–176. Springer-Verlag, August 2002.

[27] D. Cenzer and J.B. Remmel. Polynomial-time versus recursive models. *Annals of Pure and Applied Logic*, 54:17–58, 1991.

[28] K. Chakrabarty. Design on system-on-a-chip test access architecture using integer linear programming. In *Proceedings of 18th IEEE VLSI Test Symposium*, pages 127–134. IEEE Computer Society, April 2000.

[29] P.A. Cholak, R.G. Downey, and L.A. Harrington. On the orbits of computable enumerable sets. *Bulletin of Symbolic Logic*, 14(1):69–87, March 2008.

[30] P.J. Cohen. Decision procedures for real and p-adic fields. *Communications on Pure and Applied Mathematics*, 22:131–151, 1969.

[31] D. Cooper. Theorem-proving in arithmetic without multiplication. In B. Meltzer and D. Michie, editors, *Proceedings of the Seventh Annual Machine Intelligence Workshop*, volume 7 of *Machine Intelligence*, pages 91–101. Edinburgh University Press, 1972.

[32] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.

[33] G.B. Dantzig and B.C. Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

[34] C. Delhommé. Automaticité des ordinaux et des graphes homogènes. *C.R. Académie des sciences Paris, Ser. I*, 339:5–10, 2004.

[35] D.L. Dill, S. Berezin, and C. Barrett. CVC Lite: Cooperating validity checker. `http://chicory.stanford.edu/CVC`.

[36] D.P. Dubhashi. *Algorithmic Investigations in p-Adic fields*. PhD thesis, Cornell University, August 1992.

[37] S. Eilenberg. *Automata, Languages, and Machines (Vol. A)*. Academic Press, New York, 1974.

[38] D.B.A. Epstein, M.S. Paterson, G.W. Camon, D.F. Holt, S.V. Levy, and W.P. Thurston. *Word Processing in Groups*. A.K. Peters, Ltd., Natick, Massachussetts, 1992.

[39] Yu.L. Ershov. On elementary theories of local fields. *Algebra in Logika*, 4:5–30, 1965.

[40] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of 9th International Conference on Computer Aided Verification*, volume 1855 of *LNCS*, pages 232–247. Springer-Verlag, 2000.

[41] S.S. Goncharov and J.F. Knight. Computable structure and non-structure theorems. *Algebra and Logic*, 41(6):351–373, 2002.

[42] F.Q. Gouvea. *p-adic Numbers: An Introduction*. Universitext. Springer-Verlag, 2nd edition, 1997.

[43] V.S. Harizanov. Pure computable model theory. In Yu.L. Ershov, S.S. Goncharov, A. Nerode, and J.B. Remmel, editors, *Handbook of Recursive Mathematics*, volume 1, pages 3–114. North-Holland, Amsterdam, 1998.

[44] J. Harrison. Recursive pseudo well-orderings. *Transactions of the American Mathematical Society*, 131(2):526–543, 1968.

[45] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[46] K. Hensel. Über eine neue begründung der theorie der algebraischen zahlen. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 6(3):83–88, 1897.

[47] K. Hensel. Theorie der algebraischen zahlen i. Teubner, Leipzig, 1908.

[48] T. Hirst and D. Harel. Taking it to the limit: On infinite variants of NP-complete problems. *Journal of Computer and System Science*, 53:180–193, 1996.

[49] B.R. Hodgson. On direct products of automaton decidable theories. *Theoretical Computer Science*, 19:331–335, 1982.

[50] R.G. Jeroslow. *Logic-based decision support: mixed integer model formulation*. North-Holland, 1989.

[51] B. Khoussainov, J. Liu, and M. Minnes. Deciding the isomorphism problem for classes of unary automatic structures. In Preparation.

[52] B. Khoussainov, J. Liu, and M. Minnes. Unary automatic graphs: An algorithmic perspective. In M. Agrawal et al., editor, *Proceedings of 5th Conference on Theory and Applications of Models of Computation*, volume 4978 of *LNCS*, pages 548–559. Springer-Verlag, 2008.

[53] B. Khoussainov and M. Minnes. Model theoretic complexity of automatic structures (extended abstract). In M. Agrawal et al., editor, *Proceedings of 5th Conference on Theory and Applications of Models of Computation*, volume 4978 of *LNCS*, pages 520–531. Springer-Verlag, 2008.

[54] B. Khoussainov and M. Minnes. Three lectures on automatic structures. In *Proceedings of Logic Colloquium 2007*. Cambridge University Press, 2008.

[55] B. Khoussainov and M. Minnes. Model theoretic complexity of automatic structures. *Annals of Pure and Applied Logic*, To appear, 2008.

[56] B. Khoussainov and A. Nerode. Automatic presentations of structures. In D. Leivant, editor, *International Workshop on Logic and Computational Complexity*, volume 960 of *LNCS*, pages 367–392. Springer-Verlag, 1995.

[57] B. Khoussainov and A. Nerode. *Automata Theory and its Applications*. Birkhauser, Boston, Massachusetts, 2001.

[58] B. Khoussainov and A. Nerode. Open questions in the theory of automatic structures. *Bulletin of the European Association for Theoretical Computer Science*, (94):181–204, February 2008. Presented at Dagstuhl Seminar 7441.

[59] B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: Richness and limitations. In *Proceedings of 19th IEEE Symposium on Logic in Computer Science*, pages 44–53, Turku, Finland, July 2004. IEEE Computer Society.

[60] B. Khoussainov and S. Rubin. Graphs with automatic presentations over a unary alphabet. *Journal of Automata, Languages and Combinatorics*, 6(4):467–480, 2001.

[61] B. Khoussainov, S. Rubin, and F. Stephan. On automatic partial orders. In *Proceedings of 18th IEEE Symposium on Logic in Computer Science*, pages 168–177. IEEE Computer Society, June 2003.

[62] B. Khoussainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In *Proceedings of 21st International Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *LNCS*, pages 440–451. Springer-Verlag, 2004.

[63] B. Khoussainov, S. Rubin, and F. Stephan. Automatic linear orders and trees. *ACM Transactions on Computational Logic*, 6(4):675–700, 2005.

[64] B. Khoussainov and R.A. Shore. Effective model theory: The number of models and their complexity. In S.B. Cooper and J.K. Truss, editors, *Models and Computability, Invited papers from Logic Colloquium '97*,

volume 259 of *LMSLNS*, pages 193–240. Cambridge University Press, Cambridge, England, 1999.

[65] F. Klaedtke. On the automata size for Presburger arithmetic. In *Proceedings of 19th IEEE Symposium on Logic in Computer Science*, pages 110–119, Turku, Finland, July 2004. IEEE Computer Society.

[66] N. Klarlund and A. Møller. The MONA project. `http://www.brics.dk/mona/index.html`, May 2008.

[67] J.F. Knight and J. Millar. Computable structures of rank ω_1^{CK}. Submitted to Journal of Mathematical Logic; Posted on arXiv 25 Aug 2005.

[68] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proceedings of 5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society, June 1997.

[69] D. Kuske and M. Lohrey. Hamiltonicity of automatic graphs. In Preparation, 2008.

[70] J. Liu. *Automatic Structures (provisional title).* PhD thesis, University of Auckland, In process.

[71] M. Lohrey. Automatic structures of bounded degree. In M.Y. Vardi and A. Voronkov, editors, *Proceedings of 10th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, volume 2850 of *LNAI*, pages 344–358, 2003.

[72] M. Makkai. An example concerning Scott heights. *Journal of Symbolic Logic*, 46(2):301–318, June 1981.

[73] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[74] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[75] C. Morvan. On rational graphs. In J. Tiuryn, editor, *Proceedings of 3rd International Conference on Foundations of Software Science and Computation Structures*, volume 1784 of *LNCS*, pages 252–266. Springer-Verlag, 2000.

[76] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[77] M.E. Nadel. Scott sentences and admissible sets. *Annals of Mathematical Logic*, 7:267–294, 1974.

[78] A. Nerode and J.B. Remmel. Polynomial time equivalence types. In *Logic and Computation, Proceedings of a Workshop held at Carnegie Mellon University 1987*, volume 106 of *Contemporary Mathematics*, pages 221–249. American Mathematical Society, 1990.

[79] A. Nies. Describing groups. *Bulletin of Symbolic Logic*, 13(3):305–339, 2007.

[80] A. Nies and P. Semukhin. Finite automata presentable abelian groups. In S.N. Artemov and A. Nerode, editors, *Proceedings of Logical Foundations of Computer Science*, volume 4514 of *LNCS*, pages 422–436. Springer-Verlag, 2007.

[81] A. Nies and R.M. Thomas. FA presentable groups and rings. To appear in Journal of Algebra, 2008.

[82] G.P. Oliver and R.M. Thomas. Automatic presentations for finitely generated groups. In V. Diekert and B. Durand, editors, *Proceedings of 22nd International Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *LNCS*, pages 693–704. Springer-Verlag, 2005.

[83] M. Presburger. Uber die vollstandigkeit eines gewissen systems der arithmetic ganzer zahlen, in welchem die addition als einzige operation hervortritt. *Compte Rendus des Congrès des Mathématiques des pays Slavs*, 1929.

[84] W. Pugh. The Omega test: A fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, pages 102–114, August 1992.

[85] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Scociety*, 141:1–35, July 1969.

[86] C.R. Reddy and D.W. Loveland. Presburger arithemtic with bounded quantifier alternation. In *Proceedings of 10th ACM Symposium on Theory of Computing*, pages 320–325. ACM, May 1978.

[87] A. Robinson and E. Zakon. Elementary properties of ordered abelian groups. *Transactions of the American Mathematical Society*, 96(2):222–236, August 1960.

[88] H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill Book Company, 1967.

[89] J. Rotman. *An Introduction to the Theory of Groups*. Springer-Verlag, 1994.

[90] S. Rubin. *Automatic Structures*. PhD thesis, University of Auckland, 2004.

[91] S. Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14(2):169–209, June 2008.

[92] D. Scott. Logic with denumerably long formulas and finite strings of quantifiers. In J. Addison, L. Henkin, and A. Tarski, editors, *The Theory of Models*, pages 329–341. North-Holland, 1965.

[93] L. Staiger. Finite-state ω-languages. *Journal of Computer and System Science*, 27:434–448, 1983.

[94] L. Staiger and K. Wagner. Automatentheoretische und automatenfreie charakterisierungen topologischer klassen regulärer folgenmengen. *Elektron. Informationsverarb. Kybernetik*, 10(7):379–392, 1974.

[95] W. Thomas. A short introduction to infinite automata. In W. Kuich, G. Rozenberg, and A. Salomaa, editors, *Proceedings of 5th International Conference in Developments in Language Theory*, volume 2295 of *LNCS*, pages 130–144. Springer-Verlag, 2002.

[96] W. Thomas. Constructing infinite graphs with a decidable MSO-theory. In B. Rovan and P. Vojtas, editors, *Proceedings of 28th International Symposium on Mathematical Foundations of Computer Science*, volume 2747 of *LNCS*, pages 113–124. Springer-Verlag, 2003.

[97] V. Weispfenning. On the elementary theory of Hensel fields. *Annals of Mathematical Logic*, 10(1):59–93, 1976.

[98] V. Weispfenning. Quantifier elimination and decision procedures for valued fields. In G.H. Muller and M.M. Richter, editors, *Models and Sets:*

Proceedings of Logic Colloquium '83, volume 1103 of *Lecture Notes in Mathematics*, pages 419–472, Aachen, 1984. Springer-Verlag.

[99] V. Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1-2):3–27, February-April 1988.

[100] V. Weispfenning. Mixed real-integer linear quantifier elimination (extended version). In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, pages 129–136. ACM, 1999.

[101] S. Wöhrle and W. Thomas. Model checking synchronized products of infinite transition systems. In *Proceedings of 19th IEEE Symposium on Logic in Computer Science*, pages 2–11, Turku, Finland, July 2004. IEEE Computer Society.

[102] P. Wolper and B. Boigelot. An automata-theoretic approach to Presburger arithmetic constraints (extended abstract). In A. Mycroft, editor, *Proceedings of 2nd International Static Analysis Symposium*, volume 983 of *LNCS*, pages 21–32. Springer-Verlag, September 1995.

[103] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In S. Graf and M. Schwartzbach, editors, *Proceedings of 6th International Conference Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *LNCS*, pages 1–19. Springer-Verlag, March 2000.