This paper studies infinite graphs produced from a natural unfolding operation applied to finite graphs. Graphs produced via such operations are of finite degree and automatic over the unary alphabet (that is, they can be described by finite automata over unary alphabet). We investigate algorithmic properties of such unfolded graphs given their finite presentations. In particular, we ask 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 for each of these questions. For a fixed input graph, the algorithm for the first question is in constant time and the second question is decided using an automaton that recognizes reachability relation in a uniform way. Hence, we improve on previous work, in which non-elementary or non-uniform algorithms were found.

Under consideration for publication in Math. Struct. in Comp. Science

Unary Automatic Graphs: An Algorithmic Perspective

BAKHADYR KHOUSSAINOV[1], JIAMOU LIU[1] and MIA MINNES[2]

[1] *Department of Computer Science, University of Auckland, Auckland, New Zealand.*
[2] *Department of Mathematics, Cornell University, Ithaca, NY, USA.*

Received 9 April 2008

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 element 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 4.5). Therefore, we call this class of graphs *unary automatic graphs of finite degree*.

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 word automata, Büchi automata, tree automata, and Rabin automata) to represent structures and study 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 recognized by finite automata (precise definitions are in the next section). For instance, an automatic graph is one whose set of vertices and set of edges can each be recognized by finite automata. The idea of automatic structures was initially introduced by Hodgson (Hodgson 1976) and was later rediscovered by Khoussainov and Nerode (Khoussainov and Nerode 1995). 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 (Khoussainov and Nerode 1995). This result is extended by adding the $\exists^\infty$ (there are infinitely many) and $\exists^{n,m}$ (there are m many mod n) quantifiers to the first order logic (Blumensath and Grädel 2004; Khoussainov, Rubin and Stephan 2005). Blumensath and Grädel proved a logical characterization theorem stating that automatic structures are exactly those definable in the following fragment of the arithmetic $(\omega; +, |_2, \leq, 0)$, where $+$ and $\leq$ have their usual meanings and $|_2$ is a weak divisibility predicate for which $x|_2 y$ if and only if x is a power of 2 and divides y (Blumensath and Grädel 2004). 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 (Rubin 2004). Also, Khoussainov, Nies, Rubin

and Stephan have characterized the isomorphism types of automatic Boolean algebras (Khoussainov, Nies, Rubin and Stephan 2004); Thomas and Oliver have given a full description of finitely generated automatic groups (Oliver and Thomas 2005). Some of these results have direct algorithmic implications. For example, isomorphism problem for automatic well-ordered sets and Boolean algebras is decidable (Khoussainov, Nies, Rubin and Stephan 2004).

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 examples of automatic structures whose first-order theories are non-elementary (Blumensath and Grädel 2004). On the other hand, Lohrey in (Lohrey 2003) 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 $\phi(\bar{x})$ can be done in linear time.

Most of the results about automatic structures, including the ones mentioned above, demonstrate that in various concrete senses automatic structures are not complex from a logical point of view. However, this intuition can be misleading. For example, in (Khoussainov, Nies, Rubin and Stephan 2004) 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. Also, Khoussainov and Minnes (Khoussainov and Minnes) provide examples of automatic structures whose Scott ranks can be as high as possible, fully covering the interval $[1, \omega_1^{CK} + 1]$ of ordinals (where ω_1^{CK} is the first non-computable ordinal). They also show that the ordinal heights of well-founded automatic relations can be arbitrarily large ordinals below ω_1^{CK}.

In this paper, we study the class of unary automatic graphs of finite degree. Since these graphs are described by the unfolding operation (Definition 4.4) on the pair of finite graphs $(\mathcal{D}, \mathcal{F})$, we use this pair to represent the graph. The size of this pair is the sum of the sizes of the automata that represent these graphs. In the study of algorithmic properties of these graphs one directly deals with the pair $(\mathcal{D}, \mathcal{F})$. We are interested in the following natural decision problems:

— **Connectivity Problem**. Given an automatic graph $\mathcal{G}$, decide if $\mathcal{G}$ is connected.
— **Reachability Problem**. 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. In addition, we also pose the following two problems:

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

Unfortunately, for the class of automatic graphs all of the above problems are undecidable. In fact, one can provide exact bounds on this undecidability. 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 (Rubin 2004).

Since all unary automatic structures are first-order definable in $S1S$ (the monadic second-order logic of the successor function), it is not hard to prove that all the problems above are decidable (Blumensath 1999; Rubin 2004). Direct constructions using this definability in $S1S$ yield algorithms with non-elementary time since one needs to transform $S1S$ formulas into automata (Büchi 1960). However, we provide polynomial-time algorithms for solving all the above problems for this class of graphs. We now outline the rest of this paper by explaining the main results. We comment that these polynomial-time algorithms are based on deterministic input automata.

Section 2 introduces the main definitions needed, including the concept of automatic structure. Section 3 singles out unary automatic graphs and provides a characterization theorem (Theorem 3.4). Section 4 introduces unary automatic graphs of finite degree. The main result is Theorem 4.5 that explicitly provides an algorithm for building unary automatic graphs of finite degree. This theorem is used throughout the paper. Section 5 is devoted to deciding the infinite component problem. The main result is the following:

Theorem 5.1 *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 finite automaton recognizing $\mathcal{G}$.*

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

Theorem 6.1 *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 finite automaton $\mathcal{A}$ recognizing $\mathcal{G}$. In particular, when $\mathcal{A}$ is fixed, there is a constant time algorithm that decides the infinity testing problem on $\mathcal{G}$.*

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 7. This problem has been studied in (Bouajjani, Esparza and Maler 1997; Esparza, Hansel, Rossmanith and Schwoon 2000; Thomas 2002) via the class of **pushdown graphs**. A pushdown graph is the configuration space of a pushdown automaton. Unary automatic graphs are pushdown graphs (Thomas 2002). In (Bouajjani, Esparza and Maler 1997; Esparza, Hansel, Rossmanith and Schwoon 2000; Thomas 2002) it is proved that for a pushdown graph $\mathcal{G}$, given a node v, there is an automaton that recognizes all nodes reachable from v. The number of states in the automaton depends on the input node v. This result implies that there is an algorithm that decides the reachability problem on unary automatic graphs of finite degree. However, there are several issues with this algorithm. The automata constructed by the algorithm are not uniform in v in the sense that different automata are built for different input nodes v. Moreover, the automata are nondeterministic. Hence, the size of the deterministic equivalent automata is exponential in the size of the representation

of v. Section 7 provides an alternative algorithm to solve the reachability problem on unary automatic graphs of finite degree uniformly. This new algorithm constructs a deterministic automaton $\mathcal{A}_{Reach}$ that accepts the set of pairs $\{(u, v) \mid$ there is a path from u to $v\}$. The size of $\mathcal{A}_{Reach}$ only depends on the number of states of the automaton n, and constructing the automaton requires polynomial-time in n. The practical advantage of such a uniform solution is that, when $\mathcal{A}_{Reach}$ is built, deciding whether node v is reachable from u by a path takes only linear time (details are in Section 7). The main result of this section is the following:

Theorem 7.1 *Suppose $\mathcal{G}$ is a unary automatic graph of finite degree represented by deterministic 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, the last section solves the connectivity problem for $\mathcal{G}$.

Theorem 8.1 *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 finite automaton recognizing $\mathcal{G}$.*

The authors would like to thank referees for comments on improvement of this paper.

2. Preliminaries

A **finite automaton** $\mathcal{A}$ 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 table** and $F \subset S$ is the set of **final states**. A **computation** of $\mathcal{A}$ on a word $\sigma_1\sigma_2 \ldots \sigma_n$ $(\sigma_i \in \Sigma)$ is a sequence of states, say $q_0, q_1, \ldots, q_n$, such that $q_0 = \iota$ and $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$ for all $i \in \{0, 1, \ldots, n-1\}$. If $q_n \in F$, then the computation is **successful** and we say that automaton $\mathcal{A}$ **accepts** the word. The **language** accepted by the automaton $\mathcal{A}$ is the set of all words accepted by $\mathcal{A}$. In general, $D \subset \Sigma^\star$ is **FA recognizable**, or **regular**, if D is the language accepted by some finite automaton. In this paper we always assume the automata are deterministic. For two states q_0, q_1, the **distance** from q_0 to q_1 is the minimum number of transitions required for $\mathcal{A}$ to go from q_0 to q_1.

To formalize the notion of a relation being recognized by an automaton, we define synchronous n-tape automata. Such an automaton can be thought of as a one-way Turing machine with n input tapes. Each tape is semi-infinite having written on it a word in the alphabet Σ followed by a succession of $\diamond$ 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 $\diamond$ on each tape. The automaton then stops and accepts the n-tuple of words if and only if it is in a final state.

More formally, we write $\Sigma_\diamond$ for $\Sigma \cup \{\diamond\}$ where $\diamond$ is a symbol not in Σ. The **convolution** of a tuple $(w_1, \cdots, w_n) \in \Sigma^{\star n}$ is the string $\otimes(w_1, \cdots, w_n)$ of length $\max_i |w_i|$ over the alphabet $(\Sigma_\diamond)^n$ which is defined as follows: the 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 is $\diamond$ otherwise. The **convolution** of a relation $R \subset \Sigma^{\star n}$ is the relation $\otimes R \subset (\Sigma_\diamond)^{n\star}$ formed as the set of convolutions of all the tuples in R.

An n-ary relation $R \subset \Sigma^{\star n}$ is **FA recognizable**, or **regular**, if its convolution $\otimes R$ is recognizable by a finite automaton.

A **structure** $\mathcal{S}$ consists of a countable set D called the **domain** and some relations and operations on D. We may assume that $\mathcal{S}$ only contains relational predicates since operations can be replaced with their graphs. We write $\mathcal{S} = (D, R_1^D, \ldots, R_k^D, \ldots)$ where R_i^D is an n_i-ary relation on D. The relation R_i are sometimes called basic or atomic relations. We assume that the function $i \mapsto n_i$ is always a computable one. A structure $\mathcal{S}$ is **automatic over alphabet** Σ if its domain $D \subset \Sigma^\star$ is finite automaton recognizable, and there is an algorithm that for each i produces an n_i-tape automaton recognizing the relation $R_i^D \subset (\Sigma^\star)^{n_i}$. A structure is called **automatic** if it is automatic over some alphabet. If $\mathcal{B}$ is isomorphic to an automatic structure $\mathcal{S}$, then we call $\mathcal{S}$ an **automatic presentation** of $\mathcal{B}$ and say that $\mathcal{B}$ is **automatically presentable**.

An example of an automatic structure is the word structure $(\{0,1\}^\star, L, R, E, \preceq)$, where for all $x, y \in \{0,1\}^\star$, $L(x) = x0$, $R(x) = x1$, $E(x,y)$ if and only if $|x| = |y|$, and $\preceq$ is the lexicographical order. The configuration graph of any Turing machine is another example of an automatic structure. Examples of automatically presentable structures are $(\mathbb{N}, +)$, $(\mathbb{N}, \leq)$, $(\mathbb{N}, S)$, the group $(Z, +)$, the order on the rational $(Q, \leq)$, and the Boolean algebra of finite and co-finite subsets of $\mathbb{N}$. 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 $FO + \exists^\omega + \exists^{n,m}$. We will use the following theorem without explicit reference to it.

Theorem 2.1. (Khoussainov and Nerode 1995) Let $\mathcal{A}$ be an automatic structure. There exists an algorithm that, given a formula $\phi(\bar{x})$ in $FO+\exists^\omega+\exists^{n,m}$, produces an automaton that recognizes exactly those tuples $\bar{a}$ from the structure that make ϕ true. In particular, the set of all sentences of $FO + \exists^\omega + \exists^{n,m}$ which are true in $\mathcal{A}$ is decidable.

3. Unary automatic graphs

We now turn our attention to the subclass of the automatic structures which is the focus of the paper.

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

Examples of unary automatic structures are (ω, S) and $(\omega, \leq)$. Some recent work on unary automatic structures includes a characterization of unary automatic linearly ordered sets, permutation structures, graphs, and equivalence structures (Khoussainov and Rubin 2001; Blumensath 1999). For example, unary automatic linearly ordered sets are exactly those that are isomorphic to a finite sum of orders of type ω, $\omega^\star$ (the order of negative integers), and finite n.

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

We use the following example to illustrate that this class of graphs are the best possible. Consider the class of graphs with all vertices being of the form 1^*2^* for some alphabet $\Sigma = \{1, 2\}$. At first sight, graphs of this form may have an intermediate position between unary and general automatic graphs. However, the infinite grid $G_2 = \{\mathbb{N} \times \mathbb{N}, \{((i, j), (i, j + 1)) \mid i, j \in \mathbb{N}\}, \{((i, j), (i + 1, j)) \mid i, j \in \mathbb{N}\}\}$ can be coded automatically over 1^*2^* by $(i, j) \to 1^i 2^j$, and $MSO(G_2)$ is not decidable (Wöhrel and Thomas 2004). In particular, counter machines can be coded into the grid, so the reachability problem is not decidable.

Convention. To eliminate bulky exposition, we make the following assumptions in the rest of the paper.

— The automata under consideration are viewed as deterministic. Hence, when we write "automata", we mean "deterministic finite automata".
— All structures are infinite unless explicitly specified.
— The graphs are undirected. The case of directed graphs can be treated in a similar manner.

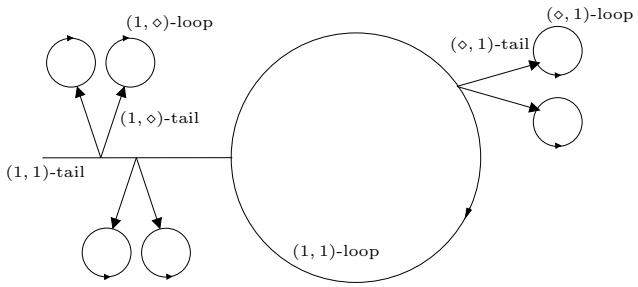

Fig. 1. A Typical Unary Graph Automaton

Let $\mathcal{G} = (V, E)$ be an automatic graph. Let $\mathcal{A}$ be an automaton recognizing E. We establish some terminology for the automaton $\mathcal{A}$. The general shape of $\mathcal{A}$ is given in Figure 1. All the states reachable from the initial state by reading inputs of type $(1, 1)$ 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 with no repetition in the middle. 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 $(1, \diamond)$ are called $(1, \diamond)$**-states**. This collection of all $(1, \diamond)$-states is also a disjoint union of a tail and a loop (see the figure), called the $(1, \diamond)$**-tail** and the $(1, \diamond)$**-loop**, respectively. The $(\diamond, 1)$**-tails** and $(\diamond, 1)$**-loops** are defined in a similar matter.

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 recognizing a binary relation, it has exactly $2p$ $(1, 1)$-states. On each of these states, there is a $(1, \diamond)$-tail and a $(\diamond, 1)$-tail of length exactly p. 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 recognizing a binary relation E on $1^\star$. 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 $8n^{2n}$. □

We can simplify the general shape of the automaton using the fact that we consider undirected graphs. Indeed, we need only consider transitions labelled by $(\diamond, 1)$. To see this, given an automaton with only $(\diamond, 1)$ transitions, to include all symmetric transitions, add a copy of each $(\diamond, 1)$ transition which is labelled with $(1, \diamond)$.

We recall a characterization theorem of unary automatic graphs from (Rubin 2004). Let $\mathcal{B} = (B, E_B)$ and $\mathcal{D} = (D, E_D)$ be finite graphs. Let R_1, R_2 be subsets of $D \times B$, and R_3, R_4 be subsets of $B \times B$. Consider the graph $\mathcal{D}$ followed by ω 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 (a^i, b^i) such that $(a, b) \in E_B$. We define the infinite graph, $unwind(\mathcal{B}, \mathcal{D}, \bar{R})$, as follows: 1) The vertex set is $D \cup B^0 \cup B^1 \cup B^2 \cup \ldots$; 2) 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 $\mathcal{A}$.

4. Unary automatic graphs of finite degree

A graph is of **finite degree** if there are at most finitely many edges from each vertex v. We call an automaton $\mathcal{A}$ recognizing a binary relation over $\{1\}$ a **one-loop automaton** if its transition diagram contains exactly one loop, the $(1, 1)$-loop. The general structure of one-loop automata is given in Figure 2.

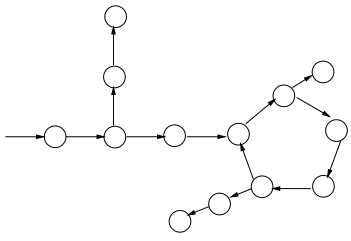

Fig. 2. One-loop automaton

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. The following is an easy proposition and we omit its proof.

Proposition 4.1. 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 $\mathcal{A}$ recognizing E. □

By Lemma 3.3, transforming a given automaton to an equivalent standard automaton may blow up the number of states exponentially. However, there is only polynomial blow up if $\mathcal{A}$ is a one-loop automaton.

Lemma 4.2. 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 $\mathcal{A}$. Let p 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 paper, 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 $\mathcal{G}_1 = (V, E_1)$ and $\mathcal{G}_2 = (V, E_2)$ (where we recall the convention that the domain of each graph is $1^\star$), we can form the **union graph** $\mathcal{G}_1 \oplus \mathcal{G}_2 = (V, E_1 \cup E_2)$ and the **intersection graph** $\mathcal{G}_1 \otimes \mathcal{G}_2 = (V, E_1 \cap E_2)$. Automatic graphs of finite degree are closed under these operations. Indeed, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be one-loop automata recognizing E_1 and E_2 with loop constants p_1 and p_2, respectively. The standard construction that builds automata for the union and intersection operations produces a one-loop automaton whose loop constant is $p_1 \cdot p_2$. We introduce another operation: consider the new graph $\mathcal{G}_1' = (V, E_1')$, where the set E_1' of edges is defined as follows; a pair $(1^n, 1^m)$ is in E' if and only if $(1^n, 1^m) \notin E$ and $|n - m| \leq p_1$. The relation E_1' is recognized by the same automaton as E_1, 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 4.3. If $\mathcal{G}_1$ and $\mathcal{G}_2$ are automatic graphs of finite degree then so are $\mathcal{G}_1 \oplus \mathcal{G}_2$, $\mathcal{G}_1 \otimes \mathcal{G}_2$, and $\mathcal{G}_1'$. □

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 4.4 (Unfolding Operation). Let $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$ and $\mathcal{F} = (V_\mathcal{F}, E_\mathcal{F})$ be finite graphs. Consider the finite sets $\Sigma_{\mathcal{D},\mathcal{F}}$ consisting of all mappings $\eta : V_\mathcal{D} \to P(V_\mathcal{F})$,

and $\Sigma_{\mathcal{F}}$ consisting of all mappings $\sigma : V_{\mathcal{F}} \to P(V_{\mathcal{F}})$. Any infinite sequence $\alpha = \eta\sigma_0\sigma_1 \ldots$ where $\eta \in \Sigma_{\mathcal{D},\mathcal{F}}$ and $\sigma_i \in \Sigma_{\mathcal{F}}$ for each i, defines the infinite graph $\mathcal{G}_\alpha = (V_\alpha, E_\alpha)$ as follows:

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

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

Figure 3 illustrates the general shape of a unary automatic graph of finite degree that is build from $\mathcal{D}$, $\mathcal{F}$, η, and σ^ω, where σ^ω is the infinite word $\sigma\sigma\sigma\cdots$.

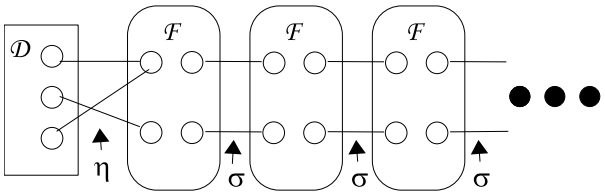

Fig. 3. Unary automatic graph of finite degree $\mathcal{G}_{\eta\sigma^\omega}$

Theorem 4.5. A graph of finite degree $\mathcal{G} = (V, E)$ possesses a unary automatic presentation if and only if there exist finite graphs $\mathcal{D}, \mathcal{F}$ and mappings $\eta : V_{\mathcal{D}} \to P(V_{\mathcal{F}})$ and $\sigma : V_{\mathcal{F}} \to P(V_{\mathcal{F}})$ such that $\mathcal{G}$ is isomorphic to $\mathcal{G}_{\eta\sigma^\omega}$.

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

Conversely, consider the graph $\mathcal{G}_{\eta\sigma^\omega}$ for some $\eta \in \Sigma_{\mathcal{D}}$ and $\sigma \in \Sigma_{\mathcal{F}}$. Assume that $V_{\mathcal{D}} = \{q_0, \ldots, q_{\ell-1}\}$, $V_{\mathcal{F}} = \{q'_0, \ldots, q'_{p-1}\}$. A one-loop automaton $\mathcal{A}$ recognizing the edge relation of $\mathcal{G}_{\eta\sigma^\omega}$ is constructed as follows. The $(1, 1)$-tail of the automaton is formed by $\{q_0, \ldots, q_{\ell-1}\}$ and the $(1, 1)$-loop is formed by $\{q'_0, \ldots, q'_{p-1}\}$, both in natural order. The

initial state is q_0. If for some $i < j$, $\{q_i, q_j\} \in E_{\mathcal{D}}$, then put a final state q_f on the $(\diamond, 1)$-tail starting from q_i such that the distance from q_i to q_f is $j - i$. If $q'_j \in \eta(q_i)$, then repeat the process but make the corresponding distance $p + j - i$. The set of edges $E_{\mathcal{F}}$ and mapping σ are treated in a similar manner by putting final states on the $(\diamond, 1)$-tails from the $(1, 1)$-loop.

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

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

Corollary 4.6. 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, $|V_{\mathcal{F}}| = |V_{\mathcal{D}}| = p$. □

5. Deciding the infinite component problem

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

Theorem 5.1. 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 recognizing $\mathcal{G}$.

By Theorem 4.5, let $\mathcal{G} = \mathcal{G}_{\eta\sigma^\omega}$. We observe that it is sufficient to consider the case in which $\mathcal{D} = \emptyset$ (hence $\mathcal{G} = \mathcal{G}_{\sigma^\omega}$) since $\mathcal{G}_{\eta\sigma^\omega}$ has an infinite component if and only if $\mathcal{G}_{\sigma^\omega}$ has one.

Let $\mathcal{F}^i$ be the i^{th} copy of $\mathcal{F}$ in $\mathcal{G}$. Let x^i be the copy of vertex x in $\mathcal{F}^i$. We construct a finite directed graph $\mathcal{F}^\sigma = (V^\sigma, E^\sigma)$ as follows. Each node in V^σ represents a distinct connected component in $\mathcal{F}$. For simplicity, we assume that $|V^\sigma| = |V_{\mathcal{F}}|$ and hence use x to denote its own component in $\mathcal{F}$. The case in which $|V^\sigma| < |V_{\mathcal{F}}|$ can be treated in a similar way. For $x, y \in V_{\mathcal{F}}$, put $(x, y) \in E^\sigma$ if and only if $y' \in \sigma(x')$ for some x' and y' that are in the same component as x and y, respectively. Constructing $\mathcal{F}^\sigma$ requires finding connected components of $\mathcal{F}$ hence takes time $O(p^2)$. To prove the above theorem, we make essential use of the following definition. See also (Hell and Nešetřil 2004).

Definition 5.2. An ***oriented walk*** in a directed graph G is a subgraph $\mathcal{P}$ of G that consists of a sequence of nodes $v_0, ..., v_k$ such that for $1 \leq i \leq k$, either (v_{i-1}, v_i) or (v_i, v_{i-1}) is an arc in G, and for each $0 \leq i \leq k$, exactly one of (v_{i-1}, v_i) and (v_i, v_{i-1}) belongs to $\mathcal{P}$. An oriented walk is an ***oriented cycle*** if $v_0 = v_k$ and there are no repeated nodes in $v_1, ..., v_k$.

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

Lemma 5.3. There is an infinite component in $\mathcal{G}$ if and only if there is an oriented cycle in $\mathcal{F}^\sigma$ such that the net length of the cycle is positive.

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

Conversely, suppose there is an infinite component D in $\mathcal{G}$. Since $\mathcal{F}$ is finite, there must be some x in $V_\mathcal{F}$ such that there are infinitely many copies of x in D. Let x^i and x^j be two copies of x in D such that $i < j$. Consider a path between x^i and x^j. We can assume that on this path there is at most one copy of any 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 5.1 By the equivalence in Lemma 5.3, it suffices to provide an algorithm that decides if $\mathcal{F}^\sigma$ contains an oriented cycle with positive net length. Notice that the existence of an oriented cycle with positive net length is equivalent to the existence of an oriented cycle with negative net length. Therefore, we give an algorithm which finds oriented cycles with non-zero net length.

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

`ALG:Oriented-Cycle`

1 Pick node $x \in \mathcal{F}^\sigma$ for which a queue has not been built yet. Initially the queue Q_x is empty. Let $d(x) = 0$, and put x into the queue. Mark x as *unprocessed*. If queues have been built for each $x \in \mathcal{F}^\sigma$, stop the process and return *NO*.

2 Let y be the first *unprocessed* node in Q_x. If there are no *unprocessed* nodes in Q_x, return to (1).

3 For each of the nodes z in the set $\{z \mid (y, z) \in E^\sigma$ or $(z, y) \in E^\sigma\}$, do the following.

   (a) If $(y, z) \in E^\sigma$, set $d'(z) = d(y) + 1$; if $(z, y) \in E^\sigma$, set $d'(z) = d(y) - 1$. (If both hold, do steps (a), (b), (c) first for (z, y) and then for (y, z).)

   (b) If $z \notin Q_x$, then set $d(z) = d'(z)$, put z into Q_x, and mark z as *unprocessed*.

   (c) If $z \in Q_x$ then

      i if $d(z) = d'(z)$, move to next z,

      ii if $d(z) \neq d'(z)$, stop the process and return *YES*.

4 Mark y as *processed* and go back to (2).

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

We claim that the algorithm returns *YES* if and only if there is an oriented cycle in $\mathcal{F}^\sigma$ with non-zero net length. Suppose the algorithm 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 non-zero net length, then it is our required oriented cycle; otherwise, we disregard the part between the two occurrences of z and make the oriented walk shorter without altering its net length.

Conversely, suppose there is an oriented cycle $\mathcal{P} = x_0, \ldots, x_m$ of non-zero net length where $x_0 = x_m$. However, we assume for a contradiction that the algorithm returns *NO*. Consider how the algorithm acts when we pick x_0 at step (1). For each $i \in \{0, 1, \ldots, m\}$, one can prove the following statements by induction on i.

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

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

In summary, the following algorithm solves the infinite component problem. Suppose we are given an automaton (with loop constant p) which recognizes the unary automatic graph of finite degree $\mathcal{G}$. Recall that p is also the cardinality of $V_\mathcal{F}$. We first compute $\mathcal{F}^\sigma$, in time $O(p^2)$. Then we run `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 $\mathcal{F}^\sigma$ contains p number of nodes, this takes time $O(p^3)$.

Note that Lemma 5.3 holds for the case when $|V_\mathcal{F}| > |\mathcal{F}/\sim_{comp}|$. Therefore the algorithm above can be slightly modified to apply to this case as well. □

6. 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. We prove the following theorem.

Theorem 6.1. The infinity testing problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(p^3)$, where p is the loop constant of the automaton $\mathcal{A}$ recognizing $\mathcal{G}$. In particular, when $\mathcal{A}$ is fixed, there is a constant time algorithm that decides the infinity testing problem on $\mathcal{G}$.

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

Lemma 6.2. If x^i is connected to some y^j such that $|j-i| > p$, then x^i is in an infinite component.

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

Let $i' = min\{p, i\}$. To decide 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 as follows:

`ALG: FiniteReach`

1 Let $i' = \min\{p, i\}$.
2 Initialize the queue Q to be empty. Put the pair $(x, 0)$ into Q and mark it as *unprocessed*.
3 If there are no *unprocessed* pairs in Q, stop the process. Otherwise, let (y, d) be the first *unprocessed* pair. For arcs e of the form (y, z) or (z, y) in E^σ, do the following.
   (a) If e is of the form (y, z), let $d' = d + 1$; if e is of the form (z, y), let $d' = d - 1$.
   (b) If $-i' \leq d' \leq p$ and (z, d') is not in Q, then put (z, d') into Q and mark (z, d') as *unprocessed*.
4 Mark (y, d) as *processed*, and go to (2).

Note that any y^j is reachable from x^i on the graph $\mathcal{G}$ restricted on $\mathcal{F}^{i-i'}, \ldots, \mathcal{F}^{i+p}$ if and only if after running `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 is $O(p^3)$. Let $B = \{y \mid (y, p) \in Q\}$.

Lemma 6.3. Let $x \in V_{\mathcal{F}}$. x^i is in an infinite component if and only if $B \neq \emptyset$.

Proof. Suppose a vertex $y \in B$, then there is a path from x^i to y^{i+p}. By Lemma 6.2, x^i is in an infinite component. Conversely, if x^i is in an infinite component, then there must be some vertices in $\mathcal{F}^{i+p}$ reachable from x^i. Take a path from x^i to a vertex y^{i+p} such that y^{i+p} is the first vertex in $\mathcal{F}^{i+p}$ appearing on this path. Then $y \in B$. □

Proof of Theorem 6.1 We assume the input vertex x^i is given by tuple (x, i). The above lemma suggests a simple algorithm to check if x^i is in an infinite component.

`ALG: InfiniteTest`

1 Run `FiniteReach` on vertex x^i, computing the set B while building the queue Q.
2 For every $y \in B$, check if there is edge $(y, z) \in E^\sigma$. Return YES if one such edge is found; otherwise, return No.

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

7. Deciding the reachability problem

Suppose $\mathcal{G}$ is a unary automatic graph of finite degree represented by an automaton with loop constant p. The **reachability problem** on $\mathcal{G}$ is formulated as: 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 7.1. Suppose $\mathcal{G}$ is a unary automatic graph of finite degree represented by an automaton $\mathcal{A}$ of loop constant p. 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 $O(|u| + |v| + p^4)$.

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

Since, by Theorem 6.1, 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 when the input x^i is in a finite component. By Lemma 6.2, 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 7.2. 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. Line(3b) in the algorithm is changed to the following:

(3b) 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 $\mathit{Reach}(x) = \{y \mid (y, 0) \in Q\}$. Intuitively, we can think of the algorithm as a breadth first search through $\mathcal{F}^0 \cup \cdots \cup \mathcal{F}^{2p}$ which originates at x^p. Therefore, $y \in \mathit{Reach}(x)$ 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 7.3. 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 $y \in \mathit{Reach}(x)$.

Proof. Suppose y^i is in an infinite component and $y \in \mathit{Reach}(x)$. 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 $y \in \mathit{Reach}(x)$, 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 $\{x^{p+rm} \mid m \in \omega\}$ are in the same component. Likewise, there is an $r' > 0$ such that all vertices in $\{y^{p+r'm} \mid m \in \omega\}$ are in the same component. Consider $x^{p+rr'}$ and $y^{p+rr'}$. Analogous to the path P_1, there is a path P_1' from $x^{p+rr'}$ to $z^{\ell+rr'}$. Similarly, there is a path P_2' from $z^{\ell+rr'}$ to $y^{p+rr'}$. We describe another path P' from x^p to y^p as follows. P' first goes from x^p to $x^{p+rr'}$, then goes along $P_1'P_2'$ 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 $y \in \mathit{Reach}(x)$. 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, $y \in \mathit{Reach}(x)$. On the other hand, suppose no such path exists. Since x^i and y^i are in the same component, there is some path P from x^i to y^i. Let $\ell(P)$ be the largest number such that $P \cap \mathcal{F}^{\ell(P)} \neq \emptyset$. Let $\ell'(P)$ be the least number such that $P \cap \mathcal{F}^{\ell'(P)} \neq \emptyset$. We are in one of two cases: $\ell(P) > i + p$ or $\ell'(P) < i - p$. We will prove that if $\ell(P) > i + p$ then there is a path P' from x^i to y^i such that $\ell(P') < \ell(P)$ and $\ell'(P') \geq i - p$. The case in which $\ell'(P) < i - p$ can be handled in a similar manner.

Without loss of generality, we assume $\ell'(P) = i$ since otherwise we can change the input x and make $\ell'(P) = i$. Let z be a 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. It is easy to see that $\ell(P') = \ell(P) - n$. Also since $\ell'(P) = i$, $\ell'(P') > i - p$. Therefore P' is our desired path. □

In the following, we abuse notation by using *Reach* and σ on subsets of $V_{\mathcal{F}}$. We inductively define a sequence $Cl_0(x), Cl_1(x), \ldots$ such that each $Cl_k(x)$ is a subset of $V_{\mathcal{F}}$. Let $Cl_0(x) = \mathit{Reach}(x)$ and For $k > 0$, we define $Cl_k(x) = \mathit{Reach}(\sigma(Cl_{k-1}(x)))$. The following lemma is immediate from this definition.

Lemma 7.4. Suppose x^i is in an infinite component, then x^i and y^j are in the same component if and only if y^j is also in an infinite component and $y \in Cl_{j-i}(x)$. □

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

`ALG: NaïveReach`

1 Check if each of x^i, y^j are in an infinite component of $\mathcal{G}$ (using the algorithm of Theorem 6.1).
2 If exactly one of x^i and y^j is in a finite component, then return *NO*.
3 If both x^i and y^j are in finite components, then run `FiniteReach` on input x^i and check if $(y, j-i)$ is in Q.
4 If both x^i and y^j are in infinite components, then compute $Cl_{j-i}(x)$. If $y \in Cl_{j-i}(x)$, return *YES*; otherwise, return *NO*.

We now consider the complexity of this algorithm. 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 and hence tweak the algorithm.

From Lemma 6.3, 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 component. 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 $\{x^{i+mr} \mid m \in \omega\}$ belong to the same component.

Lemma 7.5. $Cl_0(x) = Cl_r(x)$.

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 line (4) in `NaïveReach` with

(4) If x^i and y^j belong to infinite components, then compute $Cl_0(x), \ldots, Cl_{r-1}(x)$. If $y \in Cl_k(x)$ for $k < r$ such that $j - i = k \mod r$, return *YES*; otherwise, return *NO*.

Proof of Theorem 7.1 Say input vertices are given as x^i and y^j. By Lemma 7.4 and Lemma 7.5, 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 $O(i + j + p^4)$. □

Notice that, in fact, the algorithm produces a number $k < p$ such that in order to check if x^i, 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 7.6. Given a unary automatic graph of finite degree $\mathcal{G}$ 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 string 1^{ip+x} represent vertex x^i in $\mathcal{G}$. Suppose $ip + x \leq jp + y$, we construct an automaton $\mathcal{A}_{Reach}$ that accepts $(1^{ip+x}, 1^{jp+y})$ if and only if x^i and y^j are in the same component in $\mathcal{G}$.

1 $\mathcal{A}_{Reach}$ has a $(1,1)$-tail of length p^2. Let the states on the tail be $q_0, q_1, \ldots, q_{p^2-1}$, where q_0 is the initial state. These states represent 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 $r \times p$ from $q^{p^2-x}_{ip+x}$. The states on this loop are called $\check{q}^1_{ip+x}, \ldots, \check{q}^{rp}_{ip+x}$. These states represent 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 $\hat{q}^1_x, \ldots, \hat{q}^{p^2}_x$ 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 $r \times p$. We write these states as $\tilde{q}^1_x, \ldots, \tilde{q}^{rp}_x$.

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^{jp+y-x}_{ip+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^{jp+y-x}_{ip+x} final if y^{i+j} is in an infinite component and $y \in Cl_j(x)$.
   (b) Make state $\check{q}^{jp+y-x}_{ip+x}$ 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 $\hat{q}^{jp+y-x}_x$ 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 $\tilde{q}^{jp+y-x}_x$ 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 6.1.
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 7.2.
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 7.1.

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

8. Deciding the connectivity problem

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

`ALG: NaïveConnect`

1 Construct the automaton $\mathcal{A}_{Reach}$.
2 Check if all states in $\mathcal{A}_{Reach}$ are final states. If it is the case, return YES; otherwise, return NO.

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

Theorem 8.1. The connectivity problem for unary automatic graph of finite degree $\mathcal{G}$ is solved in $O(p^3)$, where p is the loop constant of the automaton recognizing $\mathcal{G}$.

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

Lemma 8.2. 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 $\{x^{i+ms} \mid m \in \omega\}$ belong to C and i is the least such number. By minimality, $i < s$. Take a walk along the path from x^{i+s} to x^i. Let y^s be the first 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 $R = \{y \in V_{\mathcal{F}} \mid (y, 0) \in Q\}$.

Lemma 8.3. 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 7.3, and show the existence of a path between x^0 to y^0 in the subgraph restricted on $\mathcal{F}^0, \ldots, \mathcal{F}^p$. Therefore $\mathcal{G}$ is not connected. Conversely, if $R = V_{\mathcal{F}}$, then every set of the form $\{y \in V_{\mathcal{F}} \mid (y, i) \in Q\}$ for $i \geq 0$ equals $V_{\mathcal{F}}$. By Lemma 8.2, all vertices are in the same component. □

Proof of Theorem 8.1 By the above lemma the following algorithm decides the connectivity problem on G:

`ALG: Connectivity`

1 Use the algorithm proposed by Theorem 5.1 to decide if there is an infinite component in G. If there is no infinite component, then stop and return NO.

2 Pick an arbitrary $x \in V_{\mathcal{F}}$, run `FiniteReach` on x^0 to compute the queue Q.
3 Let $C = \{y \mid (y, 0) \in Q\}$. If $C = V_{\mathcal{F}}$, return YES; otherwise, return NO.

Solving the infinite component problem takes $O(p^3)$ by Theorem 5.1. Running algorithm `FiniteReach` also takes $O(p^3)$. Therefore `Connectivity` takes $O(p^3)$. □

9. Conclusion

In this paper 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. In our future work we plan to improve these algorithms for other stronger classes of unary automatic graphs. We also point out that there are many other algorithmic problems for finite graphs that can be studied for the class of unary automatic graphs. These, for example, may concern finding spanning trees for automatic graphs, studying the isomorphism problems, and other related issues.

References

Blumensath, A.(1999) Automatic Structures. Diploma Thesis, RWTH Aachen.

Blumensath, A. and Grädel, E.(2004) Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, **37**, 642–674.

Bouajjani, A., Esparza, J. and Maler, O. (1997) Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of CONCUR'97, LNCS* **1243**, 135–150. Springer-Verlag.

Büchi, J. R. (1960) On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, A. Tarski (editors), *Proc. International Congress on Logic, Methodology and Philosophy of Science*, 1–11. Stanford University Press.

Caucal, D. (2002) On infinite graphs having a decidable monadic theory. In K. Diks, W. Rytter (editors), *Proc. 27th MFCS, LNCS* **2420**, 165–176. Springer-Verlag.

Esparza, J., Hansel, D., Rossmanith, P. and Schwoon, S. (2000) Efficient algorithms for model checking pushdown systems. In *Proc. CAV 2000, LNCS* **1855**, 232–247. Springer-Verlag.

Hell, P. and Nešetřil, J. (2004) *Graphs and Homomorphisms*. Oxford University Press, 2004.

Hodgson, B. R. (1976) *Théories décidables par automate fini.*. Phd thesis, University of Montréal.

Khoussainov, B. and Minnes, M. (2008) Automatic structures and their complexity.(Extended Abstract). In *Proc. TAMC'08*, to appear.

Khoussainov, B. and Nerode, A. (1995) Automatic presentation of structures. *LNCS* **960**, 367–392. Springer-Verlag.

Khoussainov, B., Nies, A., Rubin, S. and Stephan, F. (2004) Automatic structures: richness and limitations. In *Proc. 19th LICS*, 44–53.

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

Khoussainov, B., Rubin, S. and Stephan, F, (2005) Automatic linear orders and trees. *ACM Trans. Comput. Log. 6* (4), 675–700.

Libkin, L. (2004) *Elements of finite model theory*. Springer-Verlag.

Lohrey, M. (2003) Automatic structures of bounded degree. In *Proc. 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), LNAI* **2850**, 344–358.

Oliver, G. P. and Thomas, R. M. (2005) Automatic presentations for finitely generated groups. In V. Diekert and B. Durand (editors), *Proc. 22nd STACS, LNCS* **3404**, 693–704. Springer-Verlag.

Rabin, M. O.(1969) Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.* **141**, 1–35.

Rubin, S. (2004) *Automatic Structures.* PhD Thesis, University of Auckland.

Thomas, W. (2002) A short introduction to infinite automata. In *Proceedings of the 5th International Conference Development in Language Theory, LNCS* **2295**, 130–144. Springer-Verlag.

Wöhrle, S. and Thomas, W. (2004) Model Checking Synchronized Products of Infinite Transition Systems. In *Proc. 19th Annual IEEE Symposium on Logic in Computer Science (LICS'04)*, 2–11.