

# UNARY AUTOMATIC GRAPHS: AN ALGORITHMIC PERSPECTIVE

BAKHADYR KHOUSSAINOV<sup>1</sup>, JIAMOU LIU<sup>1</sup>, MIA MINNES<sup>2</sup>

**ABSTRACT** Given a graph, it is natural to ask whether it is connected, whether it contains an infinite component, and whether two given points in the graph are reachable from one another. For infinite graphs, in general, these questions are undecidable. In this paper, we study a subclass of infinite graphs: the locally finite unary automatic graphs. We give polynomial time algorithms for each of the questions mentioned above. This improves upon previous work, in which non-elementary or exponential time algorithms were found.

**KEYWORDS** Automata, automatic structures, finite directed graphs, algorithms for infinite graphs.

## 1. INTRODUCTION

In recent years there has been increasing interest in the study of structures that are presented by automata. The underlying idea in this line of research consists of using automata to represent structures and then to study logical and algorithmic consequences of such presentations. A structure is *automatic* if the domain and all the basic relations are recognized by finite automata. For instance, a graph  $(V; E)$  is *automatic* if the sets of vertices  $V$  and edges  $E$  are recognized by finite automata (precise definitions are in the next section). 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 [7]. When both a first-order formula  $\phi(\bar{x})$  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. There are also descriptions of automatic linear orders and trees in model theoretic terms such as Cantor-Bendixson ranks [13], [10]. Moreover, Khoussainov, Nies, Rubin and Stephan have characterized the isomorphism types of automatic Boolean algebras [8]. In particular, these characterizations imply that the isomorphism problems for automatic well-ordered sets and Boolean algebras are decidable [13]. In [12], Thomas and Oliver give a full description of finitely generated automatic groups.

The computational complexity of the first-order theories of automatic structures has been extensively studied. For example, Grädel and Blumensath constructed examples of automatic structures whose first-order theories are non-elementary (see [2]). Kuske and Lohrey, on the other hand, proved that the first-order theory of any automatic graph of bounded degree is elementary (in [11]). Other results about automatic structures demonstrate their potentially high complexity in various logical terms. In [8] it is shown that the isomorphism problem for automatic structures is  $\Sigma_1^1$ -complete. This tells us that there is no hope for a good description of the isomorphism types of automatic structures. Moreover, there are examples in [6] of automatic structures whose Scott ranks fully cover the interval  $[1, \omega_1^{CK} + 1]$  of ordinals (where  $\omega_1^{CK}$  is the first non-computable ordinal). Also in [6], well-founded automatic relations are built whose heights are arbitrarily large computable ordinals.

This paper studies automatic graphs  $\mathcal{G} = (V; E)$ . The pair of automata recognizing  $V$  and  $E$  is called a **representation** of  $\mathcal{G}$ . The **size** of the representation is the sum of the sizes of the automata. We are interested in the following 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  $y$ .

For the class of finite graphs, these two problems can be solved in linear time. However, for infinite graphs, much more work is needed to investigate these problems. In this paper, we show

---

<sup>1</sup>University of Auckland; [bm@cs.auckland.ac.nz](mailto:bm@cs.auckland.ac.nz), [jliu036@ec.auckland.ac.nz](mailto:jliu036@ec.auckland.ac.nz)

<sup>2</sup>Cornell University; [minnes@math.cornell.edu](mailto:minnes@math.cornell.edu)

that for a particular class of infinite automatic graphs, much of the information can be captured by finite directed graphs. Hence, we can use algorithms which are based on breadth-first search of finite graphs to solve problems for those infinite graphs.

In addition, we pose the following two decision problems, which are nontrivial only in the case of infinite graphs.

- **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 these problems are undecidable (see [13]). The goal of this paper is to investigate these four problems for a special class of automatic graphs: locally finite graphs which are automatic over the unary alphabet. 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 ([1], [13]). Direct constructions using this definability in  $S1S$  yield algorithms with non-elementary time complexity, since one needs to transform  $S1S$  formulas into automata ([4]). However, we provide polynomial time algorithms which solve all the above problems for this class of graphs. Our algorithms are based on a characterization we give for the isomorphism types of locally finite unary automatic graphs. We now outline the rest of the paper.

Section 2 introduces the main definitions needed, including the concept of automatic graphs, and recalls a characterization theorem (Theorem 2.2) for unary automatic graphs. Section 3 introduces locally finite automatic graphs. The central theorem (Theorem 3.2) explicitly provides an algorithm for building locally finite unary automatic graphs. This theorem is used throughout the paper. Section 4 is devoted to deciding the infinite component problem. The section introduces the main technical concept of edge path for finite directed graphs. The main result is:

**Theorem 4.1** *The infinite component problem for locally finite unary automatic graph  $\mathcal{G}$  is solved in  $O(n^{\frac{3}{2}})$ , where  $n$  is the number of states of the unary automaton for the edge relation in  $\mathcal{G}$ .*

Section 5 solves the infinity testing problem by exploiting the concept of edge path:

**Theorem 5.1** *There is a time  $O(n^{\frac{5}{2}})$  algorithm which, given a unary automaton  $\mathcal{A}$  of size  $n$  representing a locally finite graph  $\mathcal{G}$ , and given a vertex  $x$ , decides whether  $x$  forms an infinite component in  $\mathcal{G}$ . In particular, when  $\mathcal{A}$  is fixed, there is a constant time algorithm to decide the infinity testing problem on its corresponding graph  $\mathcal{G}$ .*

In the case where  $\mathcal{A}$  is fixed, the value of the constant for the constant time algorithm is polynomial in the number of states of the automaton representing the graph.

The reachability problem is addressed in Section 6. This problem has been studied in [3], [5], and [14] via pushdown graphs. A pushdown graph is the configuration space of a pushdown automaton. Unary automatic graphs are examples of pushdown graphs [14]. In [3], [5], [14] it is proved that for a given node  $v$  in a pushdown graph, there is an automaton that recognizes all nodes reachable from  $v$ . The number of states in the automaton depends on the size of  $v$ . Thus, there is an algorithm that decides the reachability problem on these locally finite unary automatic graphs. However, this algorithm encounters several obstacles. The automata constructed by the algorithm are not uniform since different automata are built for different vertices  $v$ . Also, the automata are nondeterministic. Hence, the size of the equivalent deterministic automata is exponential in the size of  $v$ . Therefore, this algorithm solves the reachability problem in exponential time. Section 6 provides an alternative, polynomial time, algorithm which solves the reachability problem.

**Theorem 6.1** *Suppose  $\mathcal{G}$  is a locally finite graph represented by unary automaton  $\mathcal{A}$  of size  $n$ . There exists a polynomial-time algorithm that solves the reachability problem on  $\mathcal{G}$ . For inputs  $v, w$ , the running time of the algorithm is  $O(|v| + |w| + n^{\frac{5}{2}})$ .*

The last result of Section 6 follows from the algorithm for the reachability problem and solves the connectivity problem for  $\mathcal{G}$ .

**Theorem 6.8** *Given a locally finite graph  $\mathcal{G}$  represented by a unary automaton of size  $n$  there is a time  $O(n^3)$  algorithm which checks if  $\mathcal{G}$  is connected.*

## 2. PRELIMINARIES

A finite automaton  $\mathcal{A}$  over an alphabet  $\Sigma$  is a tuple  $(Q, \iota, \Delta, F)$ , where  $Q$  is a finite set of **states**,  $\iota \in Q$  is the **initial state**,  $\Delta \subset Q \times \Sigma \times Q$  is the **transition table** and  $F \subset Q$  is the set of **final states**. A **run** of  $\mathcal{A}$  on a word  $\sigma_1 \dots \sigma_n \in \Sigma^*$  is a sequence  $q_0, \dots, q_n$  such that  $q_0 = \iota$  and  $(q_i, \sigma_{i+1}, q_{i+1}) \in \Delta$  for all  $i \leq n - 1$ . If  $q_n \in F$ , then the run is **successful** and we say that the automaton  $\mathcal{A}$  *accepts* the word. The **language** accepted by the automaton  $\mathcal{A}$  is the set of all words accepted by  $\mathcal{A}$ . A set  $D \subset \Sigma^*$  is **finite automaton (FA) recognizable**, or **regular**, if  $D$  is the language accepted by some finite automaton.

We now consider 2-tape automata, which are one-way Turing machines with two input tapes. Each tape is semi-infinite having written on it a word from  $\Sigma^*$  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 2-tuple of words on its input tapes if it is in a final state. Formally, set  $\Sigma_\diamond = \Sigma \cup \{\diamond\}$  where  $\diamond \notin \Sigma$ . The **convolution** of a tuple  $(w_1, w_2) \in \Sigma^{*2}$  is the string  $\otimes(w_1, w_2)$  of length  $\max_i |w_i|$  over the alphabet  $(\Sigma_\diamond)^2$  which is defined as follows: the  $k^{th}$  symbol is  $(\sigma_1, \sigma_2)$  where  $\sigma_i$  is the  $k^{th}$  symbol of  $w_i$  if  $k \leq |w_i|$ , and is  $\diamond$  otherwise. The **convolution** of a relation  $E \subset \Sigma^{*2}$  is the language  $\otimes E = \{\otimes(w_1, w_2) \mid (w_1, w_2) \in E\}$ . A **2-tape automaton** on  $\Sigma$  is a finite automaton over the alphabet  $(\Sigma_\diamond)^2$ . The relation  $E \subset \Sigma^{*2}$  is **FA recognizable**, or **regular**, if  $\otimes E$  is recognizable by a 2-tape automaton.

A graph  $\mathcal{G} = (V, E)$  is **automatic** over the alphabet  $\Sigma$  if its vertex set  $V \subset \Sigma^*$  and the edge relation  $E$  are FA recognizable. An example of an automatic graph is the binary tree  $(\{0, 1\}^*; E)$ , where  $E = \{(x, y) \mid y = x0 \text{ or } y = x1\}$ . In this paper, we are interested in unary automatic graphs:

**Definition 2.1.** *A unary automatic graph is a graph  $(V, E)$  whose domain is a regular subset of  $\{1\}^*$  and whose edge relation  $E$  is regular.*

**Convention.** To eliminate bulky exposition, we make the following assumptions in the rest of the paper: 1) By “automatic graph”, we always mean “unary automatic graph”. 2) All graphs are infinite unless explicitly specified. 3) The domains of automatic graphs coincide with the set  $1^*$  of all unary strings  $\{\lambda, 1, 11, 111, \dots\}$ . All the notions and results below can be adapted to the case when the domains are regular subsets of  $1^*$ . 4) The graphs are undirected.

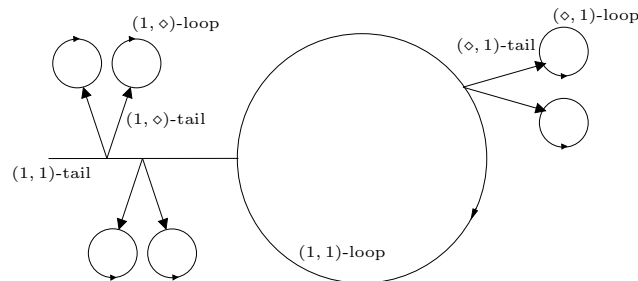


FIGURE 1. A Typical Unary Graph Automaton

Let  $\mathcal{G} = (V, E)$  be an automatic graph. Let  $\mathcal{A}$  be an  $n$  state automaton recognizing  $E$ . The general shape of  $\mathcal{A}$  is given in Figure 1. All the states reachable from the initial state by reading

input  $(1, 1)$  are called  $(1, 1)$ -**states**. 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  $q$  be a  $(1, 1)$ -state. All the states reachable from  $q$  by reading inputs  $(1, \diamond)$  are called  $(1, \diamond)$ -**states**. This collection of  $(1, \diamond)$ -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**. The  $(\diamond, 1)$ -**tails** and  $(\diamond, 1)$ -**loops** are defined in a similar way. Since we consider undirected graphs, we simplify the general shape of the automaton by only considering edges labelled by  $(\diamond, 1)$ . An automaton is **standard** if the lengths of all its loops and tails equal some number  $p$ , called the **loop constant**. Let  $\mathcal{A}$  be a unary automaton with  $n$  states. If  $\mathcal{A}$  is standard, then  $n = 8p^2$ ; otherwise, there is an equivalent standard automaton with at most  $8n^{2n}$  states.

We recall a characterization theorem of unary automatic graphs from [13]. 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  $\omega$  many copies of  $\mathcal{B}$ , ordered as  $\mathcal{B}^0, \mathcal{B}^1, \mathcal{B}^2, \dots$ . 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  $D \cup B^0 \cup B^1 \cup B^2 \cup \dots$ ; 2) The edge set contains  $E_D \cup E^0 \cup E^1 \cup \dots$  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 2.2.** [9] *A graph  $\mathcal{G}$  has a unary automaton presentation 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  $\mathcal{A}$ .*

### 3. LOCALLY FINITE UNARY AUTOMATIC GRAPHS

A graph is **locally finite** if there are 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. We will always assume that the lengths of all the tails of a one-loop automaton are not bigger than the size of its  $(1, 1)$ -loop. The following is an easy proposition:

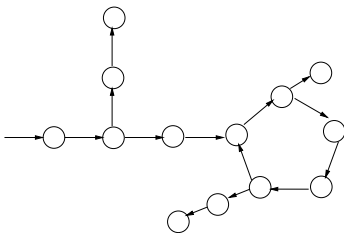


FIGURE 2. One-loop automaton

**Proposition 3.1.** *Let  $\mathcal{G} = (V, E)$  be a unary automatic graph, and let  $\mathcal{A}$  be an automaton recognizing  $E$ . The following are equivalent: 1) The number of edges emanating from each vertex  $v \in V$  is bounded. 2)  $\mathcal{G}$  is locally finite. 3)  $\mathcal{A}$  can be chosen to be a one-loop automaton.  $\square$*

We recast Theorem 2.2 for locally finite graphs. Our analysis will show that, in this case, the parameters  $\mathcal{B}$ ,  $\mathcal{D}$ , and  $\bar{R}$  can be extracted in  $O(p^2)$  time, where  $p$  is the loop constant of the one-loop automaton representing the graph. 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 all mappings  $\eta : V_{\mathcal{D}} \rightarrow P(V_{\mathcal{F}})$ , and  $\Sigma_{\mathcal{F}}$  consisting of all mappings  $\sigma : V_{\mathcal{F}} \rightarrow P(V_{\mathcal{F}})$ . Any infinite sequence  $\alpha = \eta\sigma_0\sigma_1\dots$  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:

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

Figure 3 illustrates the general shape of a locally finite unary automatic graph that is built from  $\mathcal{D}$ ,  $\mathcal{F}$ ,  $\eta$ , and  $\sigma^\omega$ , where  $\sigma^\omega$  is the infinite word  $\sigma\sigma\sigma\cdots$ .

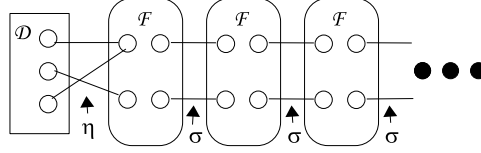


FIGURE 3. Locally finite unary automatic graph  $\mathcal{G}_{\eta\sigma^\omega}$

**Theorem 3.2.** *A locally finite graph  $\mathcal{G} = (V, E)$  possesses a unary automatic presentation if and only if there exist finite graphs  $\mathcal{D}, \mathcal{F}$  and mappings  $\eta : V_{\mathcal{D}} \rightarrow P(V_{\mathcal{F}})$  and  $\sigma : V_{\mathcal{F}} \rightarrow P(V_{\mathcal{F}})$  such that  $\mathcal{G}$  is isomorphic to  $\mathcal{G}_{\eta\sigma^\omega}$ .*

*Proof.* We outline one direction; the other direction can easily be obtained from the definition of  $\mathcal{G}_{\eta\sigma^\omega}$ . Let  $\mathcal{G} = (V, E)$  be a locally finite automatic graph. Let  $\mathcal{A}$  be an automaton for  $E$ . We can assume that  $\mathcal{A}$  is a one-loop automaton. Based on  $\mathcal{A}$ , we construct the finite graph  $\mathcal{D}$  by setting  $V_{\mathcal{D}} = \{q_0, q_1, \dots, q_{p-1}\}$ , where  $q_0, q_1, \dots, q_{p-1}$  is the sequence of nodes such that  $q_0$  is the initial state and for  $i > 0$ ,  $q_i$  is the state reached from  $q_{i-1}$  by reading  $(1, 1)$ ; 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  $j - i$ . The graph  $\mathcal{F}$  is built similarly. Also, the mapping  $\eta : V_{\mathcal{D}} \rightarrow P(V_{\mathcal{F}})$  is defined for any  $m, n \in \{0, \dots, p-1\}$  by putting  $q'_n$  in  $\eta(q_m)$  iff 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  $\sigma$  is constructed in a similar manner by reading the  $(\diamond, 1)$ -tails out of the  $(1, 1)$ -loop. The graphs  $\mathcal{G}$  and  $\mathcal{G}_{\eta\sigma^\omega}$  are isomorphic.  $\square$

#### 4. DECIDING THE INFINITE COMPONENT PROBLEM

The **infinite component problem** asks whether a given graph  $\mathcal{G}$  has an infinite component.

**Theorem 4.1.** *The infinite component problem for a locally finite unary automatic graph  $\mathcal{G}$  can be solved in  $O(p^3)$ , where  $p$  is the loop constant of the unary automaton for the edge relation in  $\mathcal{G}$ .*

By Theorem 3.2, it suffices to consider the case when  $\mathcal{G} = \mathcal{G}_{\sigma^\omega}$  (hence  $\mathcal{D} = \emptyset$ ). Let  $\mathcal{F}^i$  be the  $i^{\text{th}}$  copy of  $\mathcal{F}$  in  $\mathcal{G}$ . Let  $x^i$  be the copy of vertex  $x$  in  $\mathcal{F}^i$ . We define an equivalence relation on  $\mathcal{F}$ :  $x \sim_{\text{comp}} y$  if and only if  $x$  and  $y$  are in the same component in  $\mathcal{F}$ . Construct a finite directed graph  $\mathcal{F}^\sigma = (V^\sigma, E^\sigma)$  as follows. The set of nodes  $V^\sigma$  is  $\mathcal{F} / \sim_{\text{comp}}$ , which consists of representatives for each component in  $\mathcal{F}$ . For simplicity, in the rest of this section we assume that  $|\mathcal{F} / \sim_{\text{comp}}| = |V_{\mathcal{F}}|$  and hence use  $x$  to denote its own component in  $\mathcal{F}$ . The case in which  $|\mathcal{F} / \sim_{\text{comp}}| < |V_{\mathcal{F}}|$  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  $\sim_{\text{comp}}$ -equivalent to  $x$  and  $y$ , respectively. Constructing  $\mathcal{F}^\sigma$  requires time  $O(p^2)$ .

**Definition 4.2.** *Let  $x$  and  $y$  be nodes in  $\mathcal{F}^\sigma$ . An **edge path** from  $x$  to  $y$  is a 2-tuple  $\mathcal{P} = (\Lambda, \Gamma)$  such that  $\Lambda = (v_1, \dots, v_m)$  is a sequence of nodes in  $\mathcal{F}^\sigma$  where  $v_1 = x$  and  $v_m = y$ ; and  $\Gamma = (e_1, \dots, e_{m-1})$  is a sequence of directed edges in  $\mathcal{F}^\sigma$  such that for each  $1 \leq i < m$ , one of  $e_i = (v_i, v_{i+1})$  or  $e_i = (v_{i+1}, v_i)$  holds. An edge path is called an **edge cycle** if  $v_1 = v_m$  and there are no repeated nodes in  $\Lambda$  other than  $v_1 = v_m$ .*

In the edge path  $\mathcal{P}$ , edge  $e_i$  is called a **forward edge** if  $e_i = (v_i, v_{i+1}) \in E^\sigma$  and a **backward edge** if  $e_i = (v_{i+1}, v_i) \in E^\sigma$ . The **displacement** of  $\mathcal{P}$ , denoted  $\text{disp}(\mathcal{P})$ , is  $f - b$  where  $f$  is the

number of forward edges and  $b$  is the number of backward edges. In particular, the displacement  $\text{disp}(\mathcal{P})$  can be negative. Given an edge path  $\mathcal{P}$  of length  $m$ , for each  $1 \leq \ell \leq m$  we define  $\mathcal{P} \upharpoonright \ell$  as the edge (sub)path  $((v_1, \dots, v_\ell), (e_1, \dots, e_{\ell-1}))$ . We define the **low point** of  $\mathcal{P}$  as  $\min\{\text{disp}(\mathcal{P} \upharpoonright \ell) \mid 1 \leq \ell \leq m\}$ . The low point of edge path  $\mathcal{P}$  is at most  $\min\{0, \text{disp}(\mathcal{P})\}$ , and hence is not positive.

**Lemma 4.3.** *Let  $\mathcal{P} = (\Lambda, \Gamma)$  be an edge path from  $x$  to  $y$  whose displacement is  $d$  and whose low point is  $-\ell$ . For every  $i \geq \ell$ , the edge path  $\mathcal{P}$  defines a path  $P^i$  in  $\mathcal{G}$  from  $x^i$  to  $y^{i+d}$ . Moreover, the smallest  $j$  such that  $P^i \cap \mathcal{F}^j \neq \emptyset$  is equal to  $i - \ell$ .  $\square$*

**Lemma 4.4.** *There is an infinite component in  $\mathcal{G}$  if and only if there is an edge cycle in  $\mathcal{F}^\sigma$  with positive displacement.*

*Proof.* We prove one direction; the other is left to the reader. 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  $D$  with  $i < j$ . Consider a path between  $x^i$  and  $x^j$ . We can assume that any  $y \in V_{\mathcal{F}}$  where  $y \neq x$  appears at most once on this path (otherwise, one of the repeated nodes in the path appears infinitely often in  $D$  and has this property). By definition of  $\mathcal{G}_{\sigma^\omega}$  and  $\mathcal{F}^\sigma$ , the node  $x$  must be on an edge cycle of  $\mathcal{F}^\sigma$  with displacement  $j - i$ .  $\square$

*Proof of Theorem 4.1.* By the equivalence in Lemma 4.4, it suffices to provide an algorithm that decides if  $\mathcal{F}^\sigma$  contains an edge cycle with positive displacement. But the existence of an edge cycle with positive displacement is equivalent to the existence of an edge cycle with negative displacement. Therefore, we give an algorithm which looks for edge cycles with non-zero displacement.

**ALG:Edge-Cycle**

- (1) Pick the first node  $x \in \mathcal{F}^\sigma$  for which a queue has not been built. Initialize the queue  $Q_x$  to be empty. Let  $d(x) = 0$ , and put  $x$  into  $Q_x$  marked as *unprocessed*. If queues have been built for each  $x \in \mathcal{F}^\sigma$ , stop the process and return *NO*.
- (2) Define  $y$  to be the first *unprocessed* node in the queue  $Q_x$ . If there are no *unprocessed* nodes in  $Q_x$ , return to (1).
- (3) For each of the nodes  $z$  in the set  $\{z \mid (y, z) \in E^\sigma \text{ or } (z, y) \in E^\sigma\}$ , 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).

We claim that the algorithm returns *YES* if and only if there is an edge cycle in  $\mathcal{F}^\sigma$  with non-zero displacement. 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 edge path  $\mathcal{P}$  from  $x$  to  $z$  with displacement  $d(z)$  and there is an edge path  $\mathcal{P}'$  from  $x$  to  $z$  with displacement  $d'(z)$ . Consider the edge path  $\overleftarrow{\mathcal{P}\mathcal{P}'}$ , where  $\overleftarrow{\mathcal{P}'}$  is defined to be the edge path  $\mathcal{P}'$  in reverse direction. Clearly, this is an edge path from  $x$  to  $x$  with displacement  $d(z) - d'(z) \neq 0$ . If there are no repeated nodes in  $\overleftarrow{\mathcal{P}\mathcal{P}'}$ , then it is the required edge cycle. Otherwise, let  $y$  be a repeated node in  $\overleftarrow{\mathcal{P}\mathcal{P}'}$  satisfying the property that no nodes between the two occurrences of  $y$  are repeated. Consider the edge path between these two occurrences of  $y$ : if it has a non-zero displacement, then it is our required edge cycle; otherwise, we can make the edge path  $\overleftarrow{\mathcal{P}\mathcal{P}'}$  shorter without altering its displacement.

Conversely, suppose there is an edge cycle  $\mathcal{P}$  of non-zero displacement from a node  $x$  to itself. However, we assume for a contradiction that the algorithm returns *NO*. Let  $\Lambda$  be the sequence  $x_0, \dots, x_m$  such that  $x_0 = x_m = x$ . Consider how the algorithm behaves when we pick  $x_0$  at step (1). For each  $0 \leq i \leq m$ , the following statements hold (and can be proved by induction on  $i$ ).

- ( $\star$ )  $x_i$  gets a label  $d(x_i)$
- ( $\star\star$ )  $d(x_i)$  equals the displacement of the edge path from  $x_0$  to  $x_i$  in  $\mathcal{P}$ .

These statements suffice to yield a contradiction, and hence prove the correctness of **Edge-Cycle**.

Putting these pieces together, the following algorithm solves the infinite component problem. Suppose we are given a unary automaton (with loop constant  $p$ ) which recognizes the locally finite graph  $\mathcal{G}$ . 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 **Edge-Cycle** to decide whether  $\mathcal{F}^\sigma$  contains an edge cycle with positive displacement. For each node  $x$  in  $\mathcal{F}^\sigma$ , the run time is  $O(p^2)$ . Since  $\mathcal{F}^\sigma$  contains  $p$  nodes, this takes time  $O(p^3)$ .  $\square$

## 5. DECIDING THE INFINITY TESTING PROBLEM

The **infinity testing problem** asks for an algorithm that, given a vertex  $v$  and a locally finite unary automatic graph  $\mathcal{G}$ , decides if the vertex forms an infinite component.

**Theorem 5.1.** *There is a time  $O(p^5)$  algorithm which, given a unary automaton  $\mathcal{A}$  with loop constant  $p$  that represents a locally finite graph  $\mathcal{G}$ , and given a vertex  $x$ , decides whether  $x$  forms an infinite component in  $\mathcal{G}$ . In particular, when  $\mathcal{A}$  is fixed, there is a constant time algorithm to decide the infinity testing problem on its corresponding graph  $\mathcal{G}$ .*

The proof is based on a series of lemmas. We prove some of the lemmas but omit proofs for the easiest ones.

**Lemma 5.2.** *Let  $x \in V_{\mathcal{F}}$ . If  $x^i$  forms an infinite component of  $\mathcal{G}$  then for all  $j > 0$ ,  $x^{i+j}$  also forms an infinite component of  $\mathcal{G}$ .*  $\square$

We now explore further the connection between infinite components in  $\mathcal{G}$  and edge cycles in  $\mathcal{F}^\sigma$ . We define the set  $\mathcal{C}$  as all nodes  $x$  in  $\mathcal{F}^\sigma$  for which there exists an edge cycle from  $x$  with positive displacement and low point 0.

**Lemma 5.3.** *If  $x \in \mathcal{C}$ , then  $x^i$  is in an infinite component for all  $i \in \omega$ .*  $\square$

We call an edge path **simple** if it contains no repeated nodes. For any  $k \geq 0$ , let  $\mathcal{C}[k]$  be the set of all nodes  $x \notin \mathcal{C}[0] \cup \dots \cup \mathcal{C}[k-1]$  that can reach  $\mathcal{C}$  via a simple edge path with low point  $-k$ . Note that this definition ensures that  $\mathcal{C} \subseteq \mathcal{C}[0]$ . Moreover, since  $|\mathcal{F}^\sigma| \leq p$ , a simple edge path may have at most  $p$  steps and hence  $\mathcal{C}[k] = \emptyset$  for  $k > p-1$ .

**Lemma 5.4.** *For each vertex  $x^i$ ,  $x^i$  forms an infinite component in  $\mathcal{G}$  if and only if  $x^i \in \mathcal{C}[k]$  for some  $0 \leq k \leq \min\{i, p-1\}$ .*

*Proof.* We prove the harder direction: if  $x^i$  forms an infinite component, then there exists an edge path from  $x$  to  $\mathcal{C}$  with low point  $-k$ , where  $0 \leq k \leq \min\{i, p-1\}$ .

Let  $D$  be the infinite component of  $x^i$ . Since  $\mathcal{F}$  is finite, there must be some  $y$  in  $V_{\mathcal{F}}$  such that  $D$  contains infinitely many copies of  $y$ . Let  $y^s$  and  $y^t$  be two copies of  $y$  in  $D$  with  $s < t$ . Take a path  $P$  in  $\mathcal{G}$  between  $y^s$  and  $y^t$  such that  $P$  contains no more than one copy of each vertex in  $V_{\mathcal{F}}$  apart from  $y$ . Let  $\ell$  be the least number such that  $P \cap \mathcal{F}^\ell \neq \emptyset$ . Take a vertex  $z^\ell$  in the path  $P$ . Then  $P$  is divided into two paths  $P_1$  and  $P_2$  where  $P_1$  goes from  $y^s$  to  $z^\ell$ , and  $P_2$  goes from  $z^\ell$  to  $y^t$ . Hence there must also be a path  $P_3$  from  $y^t$  to  $z^{\ell+t-s}$ . By joining  $P_2$  and  $P_3$  together we obtain a path between  $z^\ell$  and  $z^{\ell+t-s}$ . This path defines an edge cycle in  $\mathcal{F}^\sigma$  with positive displacement and low point 0. Hence,  $z \in \mathcal{C}$ . Take a path in  $\mathcal{G}$  between  $x^i$  and a copy of  $z$  in  $D$  such that no more than one copy of each vertex in  $\mathcal{F}$  appears in the path. This path defines an edge path in  $\mathcal{F}^\sigma$  from  $x$  to  $z$  with low point less than or equal to  $\min\{i, p-1\}$ .  $\square$

**Lemma 5.5.** *Let  $\mathcal{G}$  be a locally finite graph presented by a unary automaton  $\mathcal{A}$  with loop constant  $p$ . There is an algorithm to compute  $\mathcal{C}$  for  $\mathcal{G}$  in time  $O(p^4)$ .*

*Proof.* For each  $x \in \mathcal{F}^\sigma$ , we perform a breadth-first search through  $\mathcal{F}^\sigma$  for edge paths starting at  $x$ . In order to compute the path  $\mathcal{P}$ , we put  $(y, d)$  in a queue, where  $y$  is the incremental destination of  $\mathcal{P}$  and  $d$  is its displacement. We keep track of the following properties of the pair  $(y, d)$ :

- (1)  $level(y, d)$  is the length of the edge path  $\mathcal{P}$  from  $x$  to  $y$ ; and
- (2)  $path(y, d)$  is a sequence of pairs  $(x_0, d_0) \dots (x_{level(y, d)}, d_{level(y, d)})$  representing the initial segment of  $\mathcal{P}$ . Note that  $(x_0, d_0) = (x, 0)$  and that  $(x_{level(y, d)}, d_{level(y, d)}) = (y, d)$ .

Here is the algorithm on input  $x \in \mathcal{F}^\sigma$ :

**ALG:C-Membership**

- (1) Initially the queue  $Q$  is empty. Put  $(x, 0)$  into the queue. Mark  $(x, 0)$  as *unprocessed* and set  $level(x, 0) = 0$ ,  $path(x, 0) = (x, 0)$ .
- (2) If no *unprocessed* pair is left in the queue, stop and output *NO*. Otherwise, take the first *unprocessed*  $(y, d)$  in  $Q$ .
- (3) If  $level(y, d) \geq p$ , stop and output *NO*.
- (4) For edges  $e$  of the form  $(y, z)$  or  $(z, y)$  in  $E^\sigma$  do the following:
  - (a) If  $e = (y, z)$ , set  $j = d + 1$ ; if  $e = (z, y)$ , set  $j = d - 1$ .
  - (b) If  $z = x$  and  $j > 0$ , stop the process and return *YES*.
  - (c) If  $(z, d')$  is not in  $path(y, d)$  for any  $d'$ , and if  $j \geq 0$  and  $(z, j) \notin Q$ , then we wish to process  $(z, j)$ :
    - (i) Put  $(z, j)$  into  $Q$  and mark  $(z, j)$  as *unprocessed*.
    - (ii) Set  $level(z, j) = level(y, d) + 1$  and set  $path(z, j) = path(y, d) \cdot (z, j)$ .
- (5) Mark  $(y, d)$  as *processed* and go back to (2).

We claim that **C-Membership** on input  $x$  returns *YES* if and only if  $x \in \mathcal{C}$ . Suppose that the algorithm returns *YES*. Then there exists a simple edge path  $\mathcal{P}$  from  $x$  to  $x$  with positive displacement. Let the sequence  $\Lambda$  of  $\mathcal{P}$  be  $x_0, \dots, x_m$  such that  $x_0 = x_m = x$ . The algorithm ensures that the displacement of the edge path in  $\mathcal{P}$  from  $x_0$  to each  $x_i$  is non-negative. Thus, the low point of  $\mathcal{P}$  is greater than or equal to 0. Hence,  $x \in \mathcal{C}$ .

For the other direction, suppose there exists an edge cycle  $\mathcal{P}$  from  $x$  to  $x$  of positive displacement and zero low point. Let the sequence  $\Lambda$  be  $x_0, \dots, x_m$  such that  $x_0 = x_m = x$ . Suppose the algorithm does not return *YES*. Run the algorithm from  $x_0$ . For all  $x_i$ , one can prove the following statements by induction.

- ( $\star$ ) There exists  $d_i \geq 0$  such that  $(x_i, d_i) \in Q$ .
- ( $\star\star$ )  $d_i$  equals the displacement of the edge path from  $x_0$  to  $x_i$  in  $\mathcal{P}$ .

Note that  $level(y, d) \leq p$  for all  $(y, d) \in Q$ . Moreover, every time the level is incremented by 1 the displacement is either incremented by 1 or decremented by 1, hence  $d$  must also be less than  $p$ . Therefore, the number of elements in  $Q$  is bounded above by  $p^2$  and so for each  $x \in \mathcal{F}^\sigma$ , the algorithm takes time  $O(p^3)$ . To compute  $\mathcal{C}$ , we need to run **C-Membership** on every  $x$  in  $\mathcal{F}^\sigma$ , putting  $x$  in  $\mathcal{C}$  if and only if the algorithm returns *YES*. This takes time  $O(p^4)$ .  $\square$

Using the previous lemma (Lemma 5.5), we can iteratively compute the set  $\mathcal{C}[k]$  for any  $0 \leq k \leq p - 1$  as follows. First, compute the set  $\mathcal{C}$  in time  $O(p^4)$ . For each node  $x \notin \mathcal{C}[0] \cup \dots \cup \mathcal{C}[k - 1]$ , we run operations similar to the ones described above. The differences are the following:

- (1) At step (4)(b), the process stops and returns *YES* whenever  $z \in \mathcal{C}$  and  $j \geq -k$ .
- (2) At step (4)(c), the process puts a pair  $(z, j)$  into the queue  $Q$  if  $j \geq -k$  and  $(z, j) \notin Q$ .

The proof of correctness can be done similarly to the proof above. This algorithm runs in  $O(p^4)$ .

*Proof of Theorem 5.1.* By Lemma 5.4, to check if  $x^i$  is in an infinite component, the algorithm needs to compute  $\mathcal{C}[0], \dots, \mathcal{C}[\min\{i, p - 1\}]$ . As a consequence of Lemma 5.5, this takes time  $O(p^5)$ . The algorithm then checks whether  $x^i \in \mathcal{C}[k]$  for some  $0 \leq k \leq \min\{i, p - 1\}$ .  $\square$

## 6. DECIDING THE REACHABILITY AND CONNECTIVITY PROBLEMS

Suppose  $\mathcal{G}$  is a locally finite graph presented by a unary automaton with loop constant  $p$ . We can formulate the **reachability problem** on  $\mathcal{G}$  as: given two vertices  $x^i, y^j$  in  $\mathcal{G}$ , decide if  $x^i$  and  $y^j$  are in the same component of  $\mathcal{G}$ .

**Theorem 6.1.** *Suppose  $\mathcal{G}$  is a locally finite graph presented by unary automaton  $\mathcal{A}$  with loop constant  $p$ . There exists a polynomial-time algorithm that solves the reachability problem on  $\mathcal{G}$ . For inputs  $x^i, y^j$ , the running time of the algorithm is  $O(i + j + p^5)$ .*

As before, we restrict to the case when  $\mathcal{G} = \mathcal{G}_{\sigma^\omega}$  (slight modifications are necessary for the general case). Since, by Theorem 5.1, there is an  $O(p^5)$ -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.

**Lemma 6.2.** *If  $x^i$  is in a finite component, then  $x^i$  and  $y^j$  are in the same component only if  $i - p < j < i + p$ .*  $\square$

Suppose  $x^i$  is in a finite component, 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 nodes in  $\mathcal{F}^{i-i'}, \dots, \mathcal{F}^{i+p}$ . This is sufficient by the lemma above. The algorithm is described 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 edges  $e$  of the form  $(y, z)$  or  $(z, y)$  in  $E^\sigma$ , 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).

Then,  $x^i$  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  $Q$ . The running time of **FiniteReach** is bounded by the number of edges in  $\mathcal{G}$  restricted to  $\mathcal{F}^0, \dots, \mathcal{F}^{2p}$ . Therefore the running time is  $O(p^3)$ .

**Corollary 6.3.** *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.*  $\square$

Now, suppose that  $x^i$  is in an infinite component. We start with the question: given  $y \in V_{\mathcal{F}}$ , are  $x^i$  and  $y^i$  in the same component in  $\mathcal{G}$ ? To answer this, we present an algorithm that computes all nodes  $y \in V_{\mathcal{F}}$  whose  $i^{\text{th}}$  copy lies in the same component as  $x^i$ . The algorithm is similar to **FiniteReach**, except that it does not depend on  $i$ . Line (3b) in **FiniteReach** is changed to the following:

- (3b) If  $|d'| \leq p$  and  $(z, d') \notin Q$ , then put  $(z, d')$  into  $Q$  and mark  $(z, d')$  as *unprocessed*.

We use this modified algorithm to define the set  $Reach(x) = \{y \mid (y, 0) \in Q\}$ . Intuitively, we can think of the algorithm as a breadth first search through  $\mathcal{F}^0 \cup \dots \cup \mathcal{F}^{2p}$  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  $\mathcal{F}^0 \cup \dots \cup \mathcal{F}^{2p}$ .

**Lemma 6.4.** *Suppose  $x^i$  is in an infinite component. The node  $y^i$  is in the same component as  $x^i$  if and only if  $y^i$  is also in an infinite component and  $y \in Reach(x)$ .*

*Proof.* Suppose  $y^i$  is in an infinite component and  $y \in Reach(x)$ . There is a path  $P$  in  $\mathcal{G}$  from  $x^p$  to  $y^p$ . Let  $\ell$  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 a node such that  $z^\ell \in P$ .

Then  $P$  is  $P_1P_2$  where  $P_1$  is a path from  $x^p$  to  $z^\ell$  and  $P_2$  is a path from  $z^\ell$  to  $y^p$ . By Lemma 5.2, since  $x^i$  is in an infinite component, so is  $x^p$ . There exists an  $r > 0$  such that all nodes in the set  $\{x^{p+rm} \mid m \in \omega\}$  are in the same component. Likewise, there is an  $r' > 0$  such that all nodes in  $\{y^{p+r'm} \mid m \in \omega\}$  are in the same component. Consider  $x^{p+rr'}$  and  $y^{p+rr'}$ . Analogous to the path  $P_1P_2$ , there is a path  $P'_1P'_2$  from  $x^{p+rr'}$  to  $y^{p+rr'}$ . We describe a (second) path  $P'$  from  $x^p$  to  $y^p$  as follows.  $P'$  first goes from  $x^p$  to  $x^{p+rr'}$ , then goes along  $P'_1P'_2$  from  $x^{p+rr'}$  to  $y^{p+rr'}$ , and finally goes to  $y^p$ . Notice that the least  $\ell'$  such that  $\mathcal{F}^{\ell'} \cap P' \neq \emptyset$  must be larger than  $\ell$ . 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. We want to prove that  $y \in \text{Reach}(x)$ . Let  $i' = \min\{p, i\}$ . Let  $P$  be a path in  $G$  from  $x^i$  to  $y^i$ . We will use  $P$  to construct a path which stays in  $\mathcal{F}^{i-i'} \cup \dots \cup \mathcal{F}^{i+p}$ . 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  $P \cap \mathcal{F}^{\ell'(P)} \neq \emptyset$ . If  $i - i' \leq \ell'(P)$  and  $\ell(P) \leq i + p$ , we are done. Otherwise, let  $P_1, \dots, P_k$  be a sequence of subpaths of  $P$ , each beginning and ending in  $\mathcal{F}^i$ , such that  $P = P_1 \cdots P_k$  and for each  $1 \leq j \leq k$ ,  $\ell(P_j) = i$  or  $\ell'(P_j) = i$ . We claim that each  $P_j$  can be replaced by a path  $P'_j$  with the same start and end points, and which satisfies  $i - i' \leq \ell'(P'_j) \leq \ell(P'_j) \leq i + p$ . We provide the proof for the case  $\ell(P_j) > i + p$ . We denote the initial node of  $P_j$  by  $v_j^i$  and its final node by  $w_j^i$ . Let  $z$  be a node in  $\mathcal{F}$  such that  $z^{\ell(P_j)} \in P_j$ . Then  $P_j = P_j^{(1)}P_j^{(2)}$  where  $P_j^{(1)}$  is a path from  $v_j^i$  to  $z^{\ell(P_j)}$  and  $P_j^{(2)}$  is a path from  $z^{\ell(P_j)}$  to  $w_j^i$ . Since  $\ell(P_j) > i + p$ , there must be some  $s^d$  and  $s^{d+k}$  in  $P_j^{(1)}$  such that  $k > 0$ . For the same reason, there must be some  $t^m$  and  $t^{m+n}$  in  $P_j^{(2)}$  with  $n > 0$ . Therefore,  $P_j$  contains paths between any consecutive pair of nodes in the sequence  $(v_j^i, s^d, s^{d+k}, z^{\ell(P_j)}, t^{m+n}, t^n, w_j^i)$ . Consider the sequence of nodes:

$$(v_j^i, s^d, t^{m+n-k}, t^{n-k}, s^{d-m}, s^{d+k-m}, t^n, w_j^i).$$

It is easy to check that there exists a path between each pair of consecutive nodes in the sequence. Therefore the above sequence describes a path  $P'_j$  from  $v_j^i$  to  $w_j^i$ . And,  $\ell(P'_j) = \ell(P_j) - n$ . Since  $\ell'(P_j) = i$ ,  $\ell'(P'_j) > i - p$ . Therefore,  $P'$  is our desired path, and hence  $y \in \text{Reach}(x)$ .  $\square$

We inductively define a sequence  $Cl_0(x), Cl_1(x), \dots$  such that each  $Cl_k(x)$  is a subset of  $V_{\mathcal{F}}$ . Let  $Cl_0(x) = \text{Reach}(x)$  and for  $k > 0$ , we define  $Cl_k(x) = \text{Reach}(\sigma(Cl_{k-1}(x)))$ .

**Lemma 6.5.** *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  $y \in Cl_{j-i}(x)$ .*  $\square$

Using the lemma, we construct a simple-minded algorithm for the reachability problem.

**ALG: NaïveReach**

- (1) Check if each of  $x^i, y^j$  are in an infinite component of  $\mathcal{G}$  (see Theorem 5.1).
- (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*.

In this algorithm,  $Cl_0(x)$  is computed in time  $O(p^3)$ . Given  $Cl_{k-1}(x)$ , we can compute  $Cl_k(x)$  in time  $O(p^4)$ . Therefore, the total running time of **NaïveReach** on input  $x^i, y^j$  is  $O((j - i) \cdot p^4)$ . We want to replace the multiplication with addition, and so we need to tweak the algorithm.

From Lemma 5.4,  $x^i$  is in an infinite component in  $\mathcal{G}$  if and only if there is an edge cycle  $\mathcal{C}$  with positive displacement and zero low point and which is reachable from  $x$  via a simple edge path with low point  $\geq -i$ . Now, suppose that  $x^i$  is in an infinite component. We can use the algorithm

that solves the infinity testing problem to find such an edge cycle  $C$ . Moreover, while running the algorithm, we can compute the displacement  $r$  of  $C$ . It is easy to see that all nodes in the set  $\{x^{i+mr} \mid m \in \omega\}$  belong to the same component.

**Lemma 6.6.**  $Cl_0(x) = Cl_r(x)$ . □

We give a new algorithm, **Reach**, by replacing line (4) in **NaïveReach** with

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

*Proof of Theorem 6.1.* By Lemma 6.5 and Lemma 6.6, the algorithm **Reach** returns *YES* if and only if  $x^i$  and  $y^j$  are in the same component. Since  $r \leq p$ , calculating  $Cl_0(x), \dots, Cl_{r-1}(x)$  requires time  $O(p^5)$ . Therefore the running time of **Reach** on input  $x^i, y^j$  is  $O(i + j + p^5)$ . □

Notice that, in fact, the algorithm produces a number  $k < p$  such that in order to check if  $v$  and  $w$  are in the same component, we need to test if  $|v| - |w| < p$  and if  $|v| - |w| = k$  modulo  $p$ . Thus, the algorithm takes constant time if we know the values of  $|v|, |w|$ , and  $|v| - |w|$  modulo  $p$ .

The above proof can be used to build an automaton over the unary alphabet that decides the reachability problem for locally finite automatic graphs:

**Corollary 6.7.** *Given a locally finite graph  $\mathcal{G}$  represented by a unary 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 running time of the algorithm which constructs the automaton is  $O(p^6)$ .* □

This corollary can be applied to solve the **connectivity problem** for locally finite unary automatic graphs.

**Theorem 6.8.** *Given a locally finite graph  $\mathcal{G}$  represented by a unary automaton with loop constant  $p$ , there is a time  $O(p^6)$  algorithm which checks if  $\mathcal{G}$  is connected.* □

## REFERENCES

- [1] A. Blumensath, *Automatic Structures*. Diploma Thesis, RWTH Aachen, 1999.
- [2] A. Blumensath, E. Grädel. *Finite presentations of infinite structures: Automata and interpretations*. Theory of Computing Systems, vol. 37, pp. 642-674, 2004.
- [3] A. Bouajjani, J. Esparza, and O. Maler. *Reachability analysis of pushdown automata: Application to model-checking*. In Proceedings of CONCUR'97, LNCS 1243, pages 135-150, 1997.
- [4] J. R. Büchi, *On a decision method in restricted second-order arithmetic*. Proc. International Congress on Logic, Methodology and Philosophy of Science (E. Nagel, P. Suppes, A. Tarski, Eds.), Stanford University Press, 1-11, 1960.
- [5] J. Esparza, D. Hansel, P. Rossmanith, S. Schwoon, *Efficient algorithms for model checking pushdown systems*, Proc. CAV 2000, LNCS 1855, Springer-Verlag, 232-247, 2000.
- [6] B. Khoussainov, M. Minnes, *Automatic structures and their complexity*. In preparation.
- [7] B. Khoussainov, A. Nerode, *Automatic presentation of structures*. Lecture Notes in Computer Science, 960; 367-392, 1995.
- [8] B. Khoussainov, A. Nies, S. Rubin, F. Stephan, *Automatic structures: richness and limitations*. In Proc. 19th LICS: 44-53, 2004.
- [9] B. Khoussainov, S. Rubin, *Graphs with automatic presentations over a unary alphabet*. Journal of Automata, Languages and Combinatorics 6(4): 467-480, 2001.
- [10] B. Khoussainov, S. Rubin, F. Stephan, *Automatic linear orders and trees*. ACM Trans. Comput. Log. 6(4): 675-700, 2005.
- [11] M. Lohrey, *Automatic structures of bounded degree*. Proc. 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), LNAI 2850: 344 - 358, 2003.
- [12] G. P. Oliver, R. M. Thomas, *Automatic presentations for finitely generated groups*. in Proc. 22nd STACS (V. Diekert, B. Durand, Eds.), Springer, LNCS 3404: 693 - 704, 2005.
- [13] S. Rubin, *Automatic Structures*, PhD Thesis, University of Auckland, 2004.
- [14] W. Thomas, *A short introduction to infinite automata*, In Proceedings of the 5th International Conference Development in Language Theory, Springer, LNCS 2295: 130-144, 2002.