# Unary Automatic Graphs: An Algorithmic Perspective

Bakhadyr Khoussainov[1], Jiamou Liu[1], and Mia Minnes[2]

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

**Abstract.** 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 can be described by finite automata over the 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. Hence, we improve on previous work, in which non-elementary or non-uniform algorithms were found.

## 1 Introduction

The underlying idea of automatic structures consists of using automata to represent structures and then to study the logical and algorithmic consequences of such presentations. For example, there are descriptions of automatic linear orders and trees in model theoretic terms such as Cantor-Bendixson ranks [13], [10]. Thomas and Oliver gave a full description of finitely generated automatic groups [12]. Khoussainov, Nies, Rubin and Stephan have characterized the isomorphism types of automatic Boolean algebras [8]. These results give the decidability of the isomorphism problems for automatic ordinals and Boolean algebras [13].

The complexity of the first-order theories of automatic structures has also been studied. Grädel and Blumensath constructed examples of automatic structures whose first-order theories are non-elementary [2]. Lohrey, on the other hand, proved that the first-order theory of any automatic graph of bounded degree is elementary [11]. This paper continues this line of research and investigates computational properties of unary automatic graphs of finite degree. We use a fundamental algorithmic property of automatic structures proved by Khoussainov and Nerode: the first-order theory of any automatic graph is decidable [7]. In particular, for a fixed first-order formula $\phi(\bar{x})$ and an automatic graph $\mathcal{G}$, determining if a tuple $\bar{a}$ from $\mathcal{G}$ satisfies $\phi(\bar{x})$ can be done in linear time. Refining this, we find polynomial time algorithms for natural graph theoretic questions in the class of unary automatic graphs of finite degrees. Since all such graphs can be obtained by an unfolding operation applied to finite graphs (see Theorem 2),

we measure complexity based on the input size of the finite graphs. Specifically, we are interested in the following decision problems for the graph $\mathcal{G}$ determined by the pair of finite graphs $(\mathcal{D}, \mathcal{F})$:

- **Connectivity Problem**. Is the graph $\mathcal{G}$ connected?
- **Reachability Problem**. Given vertices $x, y$, is there a path from $x$ to $y$?
- **Infinite Component Problem**. Does $\mathcal{G}$ have an infinite component?
- **Infinity Testing Problem**. Given a vertex $x$, is it in an infinite component?

For finite graphs, the first two problems can be solved in linear time and the last two have obvious answers. However, for infinite graphs, much more work is needed to investigate these problems. In the class of all automatic graphs, all of these problems are undecidable (see [13]). Since all unary automatic graphs 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]). However, the constructions which appeal to $S1S$ yield algorithms with non-elementary time complexity, since one needs to transform $S1S$ formulas into automata ([4]). The reachability 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 size of this automaton depends on the input node $v$. Moreover, the automata constructed by this algorithm are not uniform (different automata are built for different vertices $v$). It is therefore interesting to see for which classes of graphs the reachability problem has a uniform solution (an automaton that tells whether any two nodes belong to the same component). The practical advantage of a uniform solution is that, once the automaton that recognizes reachability relation is built, deciding whether node $v$ is reachable from $u$ by a path takes only linear time In this paper, we show that for unary automatic graphs of finite degree, all the problems above can be solved in polynomial time. Moreover, the reachability problem has a uniform solution.

We now outline the rest of the paper. Section 2 introduces the main definitions needed and recalls a characterization theorem (Theorem 1) for unary automatic graphs. Section 3 introduces unary automatic graphs of finite degree; Theorem 2 explicitly provides a method for building these graphs and is used throughout the paper. Section 4 and Section 5 solve the infinite component problem and infinity testing problem, respectively. For easy reference, we list the main results below. $\mathcal{G}$ is a given unary automatic graph of finite degree, $\mathcal{A}$ is the unary automaton recognizing $\mathcal{G}$, and $n$ is the number of states of $\mathcal{A}$.

**Theorem 3** *The infinite component problem for $\mathcal{G}$ is solved in $O(n^{\frac{3}{2}})$.*

**Theorem 4** *The infinity testing problem for $\mathcal{G}$ is solved in $O(n^{\frac{5}{2}})$. When $\mathcal{A}$ is fixed, a constant time algorithm decides the infinity testing problem on $\mathcal{G}$.*

Section 6 gives a polynomial time algorithm constructing uniform automata that solve the reachability problem. This algorithm also yields a solution to the connectivity problem for unary automatic graphs of finite degree.

**Theorem 5** *A polynomial time algorithm solves the reachability problem on $\mathcal{G}$. For inputs $u, v$, the running time of the algorithm is $O(|u| + |v| + n^{\frac{5}{2}})$.*
**Theorem 6** *The connectivity problem for $\mathcal{G}$ is solved in $O(n^3)$.*

## 2 Preliminaries

A **finite automaton** $\mathcal{A}$ over $\Sigma$ 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 $\sigma_1 \dots \sigma_n \in \Sigma^\star$ 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 $D \subset \Sigma^\star$ is **FA recognizable** if $D$ is the language accepted by some finite automaton. 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 $q_1$. If $|\Sigma| = 1$, we call $\mathcal{A}$ a **unary automaton**. A **2-tape automaton** is a one-way Turing machine with two semi-infinite input tapes. Each tape has written on it a word from $\Sigma^\star$ 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 $(w_1, w_2) \in \Sigma^{\star 2}$ is the string $w_1 \otimes w_2$ 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 $E \subset \Sigma^{\star 2}$ is the language $\otimes E = \{w_1 \otimes w_2 \mid (w_1, w_2) \in E\}$. The relation $E \subset \Sigma^{\star 2}$ is **FA recognizable** if $\otimes E$ is recognizable by a 2-tape automaton.

A graph $\mathcal{G} = (V, E)$ is **automatic** over $\Sigma$ if its vertex set $V \subset \Sigma^\star$ and the edge relation $E$ are FA recognizable. The binary tree $(\{0,1\}^\star, E)$, where $E = \{(x, y) \mid y = x0 \text{ or } y = x1\}$, is an automatic graph. We are interested in the following class of automatic graphs:

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

**Convention.** To eliminate bulky exposition, we fix the following assumptions: 1) By "automatic graph", we always mean "unary automatic graph". 2) All graphs are infinite unless explicitly specified otherwise. 3) The domains of automatic graphs coincide with the set $1^\star$ of all unary strings $\{\lambda, 1, 11, 111, \dots\}$. Hence, the automaton recognizing the edge relation is sufficient for describing the graph. 4) The graphs are undirected. All the notions and results below can be adapted to the case when the domains are regular subsets of $1^\star$ and when the graphs are directed without materially changing the complexity of the algorithms.

Let $\mathcal{G} = (V, E)$ be an automatic graph. Let $\mathcal{A}$ be a unary automaton recognizing $E$ with $n$ states. 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**. 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, called the $(1,1)$**-tail** and 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 $(\diamond,1)$ and $(1,1)$. An automaton is **standard** if the lengths of all its loops and tails equal some number $p$, called the **loop constant**.

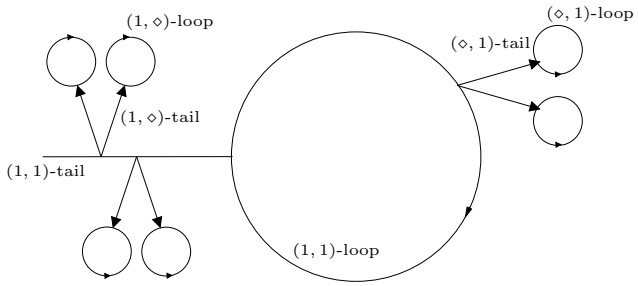


**Fig. 1.** A Typical Unary Graph Automaton

We recall a characterization theorem of unary automatic graphs from [13]. Let $\mathcal{D} = (D, E_D)$ and $\mathcal{F} = (F, E_F)$ be finite graphs. Let $R_1, R_2$ be subsets of $D \times F$, and $R_3, R_4$ be subsets of $F \times F$. Consider the graph $\mathcal{D}$ followed by $\omega$ many copies of $\mathcal{F}$, ordered as $\mathcal{F}^0, \mathcal{F}^1, \mathcal{F}^2, \ldots$. Formally, the vertex set of $\mathcal{F}^i$ is $F \times \{i\}$ and we write $f^i = (f, i)$ for $f \in F$ and $i \in \omega$. The edge set $E^i$ of $\mathcal{F}^i$ consists of all pairs $(a^i, b^i)$ such that $(a, b) \in E_F$. We define the infinite graph, $unwind(\mathcal{D}, \mathcal{F}, \bar{R})$, as follows: the vertex set is $D \cup F^0 \cup F^1 \cup F^2 \cup \ldots$; the edge set contains $E_D \cup E^0 \cup E^1 \cup \ldots$ as well as the following edges, for all $a, b \in F$, $d \in D$, 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 1.** *[9] A graph $\mathcal{G}$ has a unary automaton presentation if and only if it is isomorphic to $unwind(\mathcal{D}, \mathcal{F}, \bar{R})$ for some parameters $\mathcal{D}$, $\mathcal{F}$, and $\bar{R}$. Moreover, if $\mathcal{A}$ is a standard automaton representing $\mathcal{G}$ then the parameters $\mathcal{D}, \mathcal{F}, \bar{R}$ 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 Unary Automatic Graphs of Finite Degree

A graph is of **finite degree** if there are finitely many edges connected to each vertex $v$. A unary automaton $\mathcal{A}$ recognizing a binary relation is a **one-loop automaton** if its transition diagram contains exactly one loop, the $(1,1)$-loop. The following is an easy proposition:

**Proposition 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 unary automaton $\mathcal{A}$ recognizing $E$.* □

Each unary automaton has an equivalent standard unary automaton. In general, the standard automaton may have exponentially more states. However, if $\mathcal{A}$ is a one-loop automaton with $n$ states, the $(1,1)$-loop of the equivalent standard one-loop automaton has at most $n$ states, so the automaton itself has at most $4n^2$ states. Below, we assume the input automaton $\mathcal{A}$ is standard. Let $p$ be the loop constant of $\mathcal{A}$, then $\mathcal{A}$ has exactly $4p^2$ states. In the following, we state all results in terms of $p$ rather than $n$, the number of states of the input automaton.

**Definition 2 (Unfolding Operation).** *Let $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$ and $\mathcal{F} = (V_\mathcal{F}, E_\mathcal{F})$ be finite graphs. The finite sets $\Sigma_{\mathcal{D},\mathcal{F}}, \Sigma_\mathcal{F}$ contain all mappings $\eta : V_\mathcal{D} \to P(V_\mathcal{F})$ and $\sigma : V_\mathcal{F} \to P(V_\mathcal{F})$ (respectively). The 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 $i$ yields 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 2 illustrates the general shape of a unary automatic graph of finite degree built from $\mathcal{D}$, $\mathcal{F}$, $\eta$, and $\sigma^\omega$ ($\sigma^\omega$ is the infinite word $\sigma\sigma\sigma\cdots$). We use Definition 2 to recast Theorem 1 for graphs of finite degree. The proof is omitted.

**Theorem 2.** *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 $\mathcal{G}_{\eta\sigma^\omega}$.* □

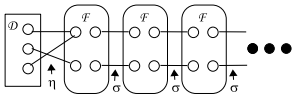


**Fig. 2.** Unary automatic graph of finite degree $\mathcal{G}_{\eta\sigma^\omega}$

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

## 4 Deciding the Infinite Component Problem

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

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

By Theorem 2, it suffices to consider the case when $\mathcal{G} = \mathcal{G}_{\sigma^\omega}$ 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 $\mathcal{G}$ and $x^i$ be the copy of vertex $x$ in $\mathcal{F}^i$. The finite directed graph $\mathcal{F}^\sigma = (V^\sigma, E^\sigma)$ is defined as follows. Nodes in $V^\sigma$ are the distinct connected components of $\mathcal{F}$. For simplicity, we assume that $|V^\sigma| = |V_\mathcal{F}|$ and use $x$ to denote its own component in $\mathcal{F}$. The case in which $|V^\sigma| < |V_\mathcal{F}|$ is similar. 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 $\mathcal{F}$ and hence takes time $O(p^2)$. To prove Theorem 3, we make essential use of the following definition which is taken from [6].

**Definition 3.** *An **oriented walk** in a directed graph $G$ is a subgraph $\mathcal{P}$ of $G$ that consists of a sequence of nodes $v_0, \ldots, v_k$ 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 $1 \leq i \leq k$, 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 $v_1, \ldots, v_k$.*

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 that the net length can be negative. Given an oriented walk $\mathcal{P} = v_0, \ldots, v_m$ , we define the **low point** of $\mathcal{P}$ as $\min\{disp(v_0 \ldots v_\ell) \mid 0 \leq \ell \leq m\}$. The low point of the oriented walk $\mathcal{P}$ is at most $\min\{0, disp(\mathcal{P})\}$, and hence is not positive. The next lemma establishes a connection between oriented walks in $\mathcal{F}^\sigma$ and paths in $G$.

**Lemma 1.** *Let $\mathcal{P}$ be an oriented walk from $x$ to $y$ whose net length is $d$ and low point is $-\ell$ . For every $i \geq \ell$, the oriented walk $\mathcal{P}$ 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 $i - \ell$.* □

**Lemma 2.** *There is an infinite component in $\mathcal{G}$ if and only if there is an oriented cycle in $\mathcal{F}^\sigma$ with positive net length.*

*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 on this path there is at most one copy of any vertex $y \in V_\mathcal{F}$ apart from $x$ (otherwise, there is another vertex in $V_\mathcal{F}$ having an infinite number of copies in the infinite component with these properties). 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 (Theorem 3).* By Lemma 2, it suffices to decide if $\mathcal{F}^\sigma$ contains an oriented cycle with positive net length. Such an oriented cycle exists if and only if there is an oriented cycle with negative net length. Therefore, the following algorithm searches for oriented cycles with non-zero net length.
`ALG:Oriented-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 there is no such $x \in \mathcal{F}^\sigma$, 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 node $z$ 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 $z \notin Q_x$, set $d(z) = d'(z)$, put $z$ into $Q_x$, and mark $z$ as *unprocessed*.
   **(c)** If $z \in Q_x$ then if $d(z) = d'(z)$, move to next $z$; 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 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 $d(z) \neq d'(z)$. Thus, 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 $d'(z)$. Let $(\mathcal{P}')^-$ be the oriented walk $\mathcal{P}'$ in reverse direction. Consider the oriented walk $\mathcal{P}(\mathcal{P}')^-$: it is an oriented walk from $x$ to $x$ with net length $d(z) - d'(z) \neq 0$. If there are no repeated nodes in $\mathcal{P}(\mathcal{P}')^-$, it is the required oriented cycle. Otherwise, let $y$ be a repeated node in $\mathcal{P}(\mathcal{P}')^-$ 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 it is our required oriented cycle and otherwise we can make the oriented walk $\mathcal{P}(\mathcal{P}')^-$ 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 $x_0 = x_m$. 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 $0 \leq i \leq m$, the following statements hold (by induction on $i$).

($\star$) $x_i$ 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 $\mathcal{P}$.

These statements suffice to yield a contradiction, and hence prove the correctness of `Oriented-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 unary automatic graph of finite degree $\mathcal{G}$. Recall that $p = |V_\mathcal{F}|$. We compute $\mathcal{F}^\sigma$ in time $O(p^2)$. Then we run `Oriented-Cycle` to decide if $\mathcal{F}^\sigma$ contains an oriented cycle with positive net length. 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 graph $\mathcal{G}$, decides if the vertex belongs to an infinite component of the graph $\mathcal{G}$.

**Theorem 4.** *The infinity testing problem for $\mathcal{G}$, a unary automatic graph of finite degree with loop constant $p$, is solved in $O(p^5)$. When $\mathcal{A}$ is fixed, there is a constant time algorithm that decides the infinity testing problem on $\mathcal{G}$.*

To prove Theorem 4 we outline several lemmas, the more difficult of which we prove. The set $\mathcal{C}$ is defined as all nodes $x$ in $\mathcal{F}^\sigma$ for which there exists an oriented cycle from $x$ with positive net length and low point 0. We call an oriented walk **simple** if it contains no repeated nodes. For any $k \geq 0$, let $\mathcal{C}[k]$ be the set of all nodes $x \notin \mathcal{C}[0] \cup \ldots \cup \mathcal{C}[k-1]$ that can reach $\mathcal{C}$ via a simple oriented walk with low point $-k$. Note that $\mathcal{C} \subseteq \mathcal{C}[0]$. Moreover, since $|\mathcal{F}^\sigma| \leq p$, a simple oriented walk may have at most $p$ steps and hence $\mathcal{C}[k] = \emptyset$ for $k > p-1$.

**Lemma 3.** *Let $x \in V_\mathcal{F}$. If $x^i$ belongs to an infinite component of $\mathcal{G}$ then for all $j > 0$, $x^{i+j}$ also belongs to an infinite component of $\mathcal{G}$.* □

**Lemma 4.** *If $x \in \mathcal{C}$, then $x^i$ is in an infinite component for all $i \in \omega$.* □

**Lemma 5.** *For each vertex $x^i$, $x^i$ belongs to an infinite component in $\mathcal{G}$ if and only if node $x \in \mathcal{C}[k]$ for some $0 \leq k \leq \min\{i, p-1\}$.*

*Proof.* We prove the harder direction: if $x^i$ is in an infinite component, there is an oriented walk 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 $y$ 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 $y$. (If there is no such path $P$, choose another vertex $y$ in $V_\mathcal{F}$ with these properties). Let $\ell$ be the least number such that $P \cap \mathcal{F}^\ell \neq \emptyset$. Let $z^\ell$ be a vertex in $P$. 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 is 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 $z^{\ell+t-s}$. We have defined an oriented cycle in $\mathcal{F}^\sigma$ with positive net length and low point 0. Hence, $z \in \mathcal{C}$. Take a path in $\mathcal{G}$ between $x^i$ and a copy of $z$ in $D$ containing no more than one copy of each vertex in $\mathcal{F}$. This is an oriented walk in $\mathcal{F}^\sigma$ from $x$ to $z$ with low point not more than $\min\{i, p-1\}$. □

**Lemma 6.** *If $\mathcal{G}$ is a unary automatic graph of finite degree presented by $\mathcal{A}$ with loop constant $p$, the set $\mathcal{C}$ for $\mathcal{G}$ can be computed in time $O(p^4)$.*

*Proof.* For each $x \in \mathcal{F}^\sigma$, do a breadth-first search through $\mathcal{F}^\sigma$ for oriented walks starting at $x$. To compute the path $\mathcal{P}$, put $(y, d)$ in a queue, where $y$ is the incremental destination of $\mathcal{P}$ and $d$ is its net length. We keep track of the following properties of the pair $(y, d)$:

1. $level(y, d)$ is the length of the oriented walk $\mathcal{P}$ from $x$ to $y$; and

2. $path(y,d)$ is a tuple of pairs $(x_0,d_0)\dots(x_{level(y,d)},d_{level(y,d)})$ coding the initial segment of $\mathcal{P}$. Note: $(x_0,d_0)=(x,0)$ and $(x_{level(y,d)},d_{level(y,d)})=(y,d)$.

Given input $x\in\mathcal{F}^\sigma$, the following algorithm checks membership in $\mathcal{C}$.

`ALG:C-Membership`

1. Put $(x,0)$ into the (initially empty) queue $Q$. 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 arcs $e$ 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 put $(z,j)$ into $Q$, mark $(z,j)$ as *unprocessed*, set $level(z,j)=level(y,d)+1$, 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 the algorithm returns *YES*. Then there is a simple oriented walk $\mathcal{P}$ from $x$ to $x$ with positive net length. Let $\mathcal{P}$ be $x_0,\dots,x_m$ such that $x_0=x_m=x$. The algorithm ensures that the net length of the (sub)oriented walk in $\mathcal{P}$ from $x_0$ to each $x_i$ is non-negative. Thus, the low point of $\mathcal{P}$ is no less than 0 and $x\in\mathcal{C}$. For the other direction, suppose $\mathcal{P}=x_0,\dots,x_m$ is an oriented cycle of positive net length and zero low point. Assume the algorithm does not return *YES*. Run the algorithm from $x_0$. For all $x_i$, the following statements hold by induction.

($\star$) There exists $d_i\geq 0$ such that $(x_i,d_i)\in Q$.
($\star\star$) $d_i$ equals the net length of the oriented walk 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 net length either goes up or down by 1, hence $d$ must also be no more than $p$. Thus, the cardinality of $Q$ 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 $\mathcal{F}^\sigma$, taking time $O(p^4)$. □

Using Lemma 6, we iteratively compute $\mathcal{C}[k]$ for any $0\leq k\leq p-1$ as follows. First, compute the set $\mathcal{C}$ in time $O(p^4)$. For each $x\notin\mathcal{C}[0]\cup\dots\cup\mathcal{C}[k-1]$ we run operations similar to the ones described above, except that at step $(4)(b)$, the process stops and returns *YES* whenever $z\in\mathcal{C}$ and $j\geq -k$, and 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 is like that of `C-Membership`. This algorithm runs in $O(p^4)$.

*Proof (Theorem 4).* We assume the input vertex $x^i$ is given by tuple $(x,i)$. By Lemma 5, 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 6, this takes time $O(p^5)$. The algorithm then checks whether $x\in\mathcal{C}[k]$ for some $0\leq k\leq\min\{i,p-1\}$. Once the sets $\mathcal{C}[0],\dots,\mathcal{C}[p-1]$ are found, checking whether $x^i$ belongs to an infinite component takes constant time. □

## 6 Deciding the Reachability and Connectivity Problems

The **reachability problem** asks whether two given vertices $u$ and $v$ in a unary automatic graph of finite degree belong to the same component.

**Theorem 5.** *Suppose $\mathcal{G}$ is a unary automatic graph of finite degree represented by unary automaton $\mathcal{A}$ of loop constant $p$. A polynomial time algorithm solves the reachability problem on $\mathcal{G}$. For inputs $x^i, y^j$, the algorithm runs in $O(i+j+p^5)$.*

We restrict to the case $\mathcal{G} = \mathcal{G}_{\sigma^\omega}$ (the general case requires few changes). The infinity testing algorithm checks if $x^i$ is in a finite component in $O(p^5)$ time, and leads to two possible cases. First, suppose that $x^i$ is in a finite component.

**Lemma 7.** *If $x^i$ is in a finite component, then $x^i$ and $y^j$ are in the same component only if $i - p < j < i + p$.* □

To check if $x^i$ and $y^j$ are in the same component, we run a breadth first search in $\mathcal{G}$ starting from $x^i$ visiting all vertices in $\mathcal{F}^{i-i'}, \ldots, \mathcal{F}^{i+p}$ ($i' = \min\{p, i\}$) .
`ALG: FiniteReach`

1. Put $(x, 0)$ into the (initially empty) queue $Q$, marked as *unprocessed*.
2. 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 $E^\sigma$:
   **(a)** If $e = (y, z)$, let $d' = d + 1$; if $e = (z, y)$, let $d' = d - 1$.
   **(b)** If $-i' \leq d' \leq p$ and $(z, d') \notin Q$, put $(z, d')$ into $Q$ marked as *unprocessed*.
3. 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 is bounded by the number of edges in $\mathcal{G}$ restricted to $\mathcal{F}^0, \ldots, \mathcal{F}^{2p}$, hence is $O(p^3)$.

**Corollary 1.** *If all components of $\mathcal{G}$ are finite and if we represent $(x^i, y^j)$ by $(x^i, y^j, j - i)$, an $O(p^3)$-algorithm checks reachability for $x^i$ and $y^j$.* □

On the other hand, suppose that $x^i$ is in an infinite component. We begin with an algorithm that computes all vertices $y \in V_{\mathcal{F}}$ whose $i^{th}$ copy lies in the same component as $x^i$. The algorithm is identical to `FiniteReach`, except that Line (2b) in `FiniteReach` is changed to the following: (**2b'**) 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 $\mathcal{F}^0 \cup \cdots \cup \mathcal{F}^{2p}$ which originates at $x^p$. Therefore, $y \in Reach(x)$ if and only if there exists a path from $x^p$ to $y^p$ in $\mathcal{G}$, restricted to $\mathcal{F}^0 \cup \cdots \cup \mathcal{F}^{2p}$.

**Lemma 8.** *If $x^i, y^i$ are both in infinite components, they are in the same component iff $y \in Reach(x)$.*

*Proof.* Assume $x^i, y^i$ are in infinite components. Suppose $y \in Reach(x)$. There is a path $P$ in $\mathcal{G}$ from $x^p$ to $y^p$. Let $\ell$ be least such that $\mathcal{F}^\ell \cap P \neq \emptyset$. If $i \geq p - \ell$, then $x^i$ and $y^i$ are in the same component. Thus, suppose $i < p - \ell$. 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 $z^\ell$ and $P_2$ is a path from $z^\ell$ to $y^p$. By Lemma 3, since $x^i$ is in an infinite component, so is $x^p$. There is $r > 0$ such that the set $\{x^{p+rm} \mid m \in \omega\}$ is contained in a single component. Likewise, there is an $r' > 0$ such that $\{y^{p+r'm} \mid m \in \omega\}$ is in one component. Consider $x^{p+rr'}$ and $y^{p+rr'}$. There is a path $P_1'P_2'$ from $x^{p+rr'}$ to $y^{p+rr'}$. A second path $P'$ from $x^p$ to $y^p$ goes from $x^p$ to $x^{p+rr'}$, then along $P_1'P_2'$ from $x^{p+rr'}$ to $y^{p+rr'}$, and finally to $y^p$. The least $\ell'$ such that $\mathcal{F}^{\ell'} \cap P' \neq \emptyset$ is larger than $\ell$. Iteratively lengthening the path between $x^p$ and $y^p$ until $i < p - \ell'$ brings us 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 $y \in Reach(x)$. Let $i' = \min\{p, i\}$. Let $P$ be a path in $G$ from $x^i$ to $y^i$. We use $P$ to construct a path which stays in $\mathcal{F}^{i-i'} \cup \cdots \cup \mathcal{F}^{i+p}$. Let $\ell(P)$ be largest such that $P \cap \mathcal{F}^{\ell(P)} \neq \emptyset$; let $\ell'(P)$ be least 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 $P_1, \ldots, P_k$ 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 $\ell'(P_j) = i$. It is not hard to see that each $P_j$ can be replaced by a path $P_j'$ with the same start and end points and which satisfies $i - i' \leq \ell'(P_j') \leq \ell(P_j') \leq i + p$. This new path witnesses that $y \in Reach(x)$. □

We inductively define a sequence $Cl_0(x), Cl_1(x), \ldots$ such that each $Cl_k(x)$ is a subset of $V_{\mathcal{F}}$. Set $Cl_0(x) = Reach(x)$. For $k > 0$, set $Cl_k(x) = Reach(\sigma(Cl_{k-1}(x)))$.

**Lemma 9.** *Suppose $j \geq i$ and $x^i, y^j$ are both in infinite components. $x^i$ and $y^j$ are in the same component if and only if $y \in Cl_{j-i}(x)$.* □

The following algorithm uses the lemma to solve the reachability problem.
`ALG: NaïveReach`

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

`Naïve Reach` computes $Cl_0(x)$ in time $O(p^3)$. Given $Cl_{k-1}(x)$, we can compute $Cl_k(x)$ in time $O(p^4)$. Hence, on input $x^i$, $y^j$, `NaïveReach` takes time $O((j - i) \cdot p^4)$. We will now improve this bound. From Lemma 5, $x^i$ is in an infinite component in $\mathcal{G}$ if and only if there is an oriented cycle $\mathcal{C}$ with positive net length, zero low point, and reachable from $x$ by a simple oriented walk with low point $\geq -i$. Assume $x^i$ is in an infinite component. The algorithm for the infinity testing problem finds such an oriented cycle $\mathcal{C}$. And, it can compute the net length $r$ of $\mathcal{C}$. All vertices in $\{x^{i+mr} \mid m \in \omega\}$ belong to the same component.

**Lemma 10.** $Cl_0(x) = Cl_r(x)$. □

We give a new algorithm, `Reach`, by replacing line (4) in `NaïveReach` with: (4') If $x^i$ and $y^j$ belong to infinite components, compute $Cl_0(x)$, ..., $Cl_{r-1}(x)$. If $y \in Cl_k(x)$ for $k < r$ with $j - i = k \mod r$, return *YES*; otherwise, return *NO*.

*Proof (Theorem 5).* By Lemma 9 and Lemma 10, `Reach` returns *YES* iff $x^i$ and $y^j$ are in the same component. Calculating $Cl_0(x), \ldots, Cl_{r-1}(x)$ requires time $O(p^5)$. Therefore the running `Reach` on $x^i$, $y^j$ takes $O(i + j + p^5)$. □

In fact, the algorithm produces $k < p$ such that to check if $x^i$, $y^j$ $(j > i)$ are in the same component, we need to test if $j - i < p$ and if $j - i = k \mod p$. If $\mathcal{G}$ is fixed, we may pre-compute $Cl_0(x), \ldots, Cl_{r_x-1}(x)$ for all $x$, so deciding if two vertices $u$, $v$ belong to the same component takes linear time. The above proof can also be used to build a unary automaton that decides reachability uniformly.

**Corollary 2.** *With $\mathcal{G}$ as above, 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 $O(p^6)$.* □

This corollary can be applied to solve the **connectivity problem**.

**Theorem 6.** *The connectivity problem for unary automatic graphs of finite degree is solved in time $O(p^6)$, where p is the loop constant of the unary automaton.* □

## 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*. Proc. CONCUR'97, LNCS 1243, 135-150, 1997.
4. J. R. Büchi, *On a decision method in restricted second-order arithmetic*. Proc. CLMPS, 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, 232-247, 2000.
6. P. Hell, J. Nešetřil, *Graphs and Homomorphisms*. Oxford University Press, 2004.
7. B. Khoussainov, A. Nerode, *Automatic presentation of structures*. LNCS 960, 367-392, 1995.
8. B. Khoussainov, A. Nies, S. Rubin, F. Stephan, *Automatic structures: richness and limitations*. Proc. LICS, 44-53, 2004.
9. B. Khoussainov, S. Rubin, *Graphs with automatic presentations over a unary alphabet*. J. 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. LPAR, LNAI 2850, 344 - 358, 2003.
12. G. P. Oliver, R. M. Thomas, *Automatic presentations for finitely generated groups*. Proc. STACS 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*, Proc. DLT, Springer, LNCS 2295, 130-144, 2002.