Isomorphism of regular trees and words[⋆]

Markus Lohrey and Christian Mathissen

Institut für Informatik, Universität Leipzig, Germany
{lohrey,mathissen}@informatik.uni-leipzig.de

Abstract. The complexity of the isomorphism problem for regular trees, regular linear orders, and regular words is analyzed. A tree is regular if it is isomorphic to the prefix order on a regular language. In case regular languages are represented by NFAs (DFAs), the isomorphism problem for regular trees turns out to be EXPTIME-complete (resp. P-complete). In case the input automata are acyclic NFAs (acyclic DFAs), the corresponding trees are (succinctly represented) finite trees, and the isomorphism problem turns out to be PSPACE-complete (resp. P-complete). A linear order is regular if it is isomorphic to the lexicographic order on a regular language. A polynomial time algorithm for the isomorphism problem for regular linear orders (and even regular words, which generalize the latter) given by DFAs is presented. This solves an open problem by Ésik and Bloom. A long version of this paper can be found in [18].

1 Introduction

Isomorphism problems for infinite but finitely presented structures are an active research topic in algorithmic model theory [1]. It is a folklore result in computable model theory that the isomorphism problem for computable structures (i.e., structures, where the domain is a computable set of natural numbers and all relations are computable too) is highly undecidable — more precisely, it is Σ_1^1-complete, i.e., complete for the first existential level of the analytical hierarchy. Khoussainov et al. proved in [15] that even for automatic structures (i.e., structures, where the domain is a regular set of words and all relations can be recognized by synchronous multitape automata), the isomorphism problem is Σ_1^1-complete. In [16], this result was further improved to automatic order trees (trees viewed as partial orders) and automatic linear orders. On the decidability side, Courcelle proved that the isomorphism problem for equational graphs is decidable [7]. Recall that a graph is equational if it is the least solution of a system of equations over the HR graph operations. We remark that Courcelle's algorithm for the isomorphism problem for equational graphs has very high complexity (it is not elementary), since it uses the decidability of monadic second-order logic on equational graphs.

In this paper, we continue the investigation of isomorphism problems for infinite but finitely presented structures at the lower end of the spectra. We focus on two very simple classes of infinite structures: *regular trees* and *regular words*; both are particular automatic structures. Recall that a countable tree is regular if it has only finitely many subtrees up to isomorphism. This definition works for ordered trees (where the children

[⋆] This work is supported by the DFG research project ALKODA.

of a node are linearly ordered) and unordered trees. An equivalent characterization in the unordered case uses regular languages: An unordered (countable) tree T is regular if and only if there is a regular language $L \subseteq \Sigma^*$ which contains the empty word and such that T is isomorphic to the tree obtained by taking the prefix order on L (ε is the root). Hence, a regular tree can be represented by a finite deterministic or nondeterministic automaton (DFA or NFA), and the isomorphism problem for regular trees becomes the following computational problem: Given two DFAs (resp., NFAs) accepting both the empty word, are the corresponding regular trees isomorphic?

It is is not difficult to prove that this problem can be solved in polynomial time if the two input automata are assumed to be DFAs; the algorithm is very similar to the well-known partition refinement algorithm for checking bisimilarity of finite state systems [14]. Hence, the isomorphism problem for regular trees that are represented by NFAs can be solved in exponential time. Our first main result states that this problem is in fact EXPTIME-complete (Thm. 3.3). The proof of the EXPTIME lower bound uses three main ingredients: (i) EXPTIME coincides with alternating polynomial space [5], (ii) a construction from [13], which reduces the evaluation problem for Boolean expressions to the isomorphism problem for (finite) trees, and (iii) a small NFA accepting all words that do *not* represent an accepting computation of a polynomial space machine [23].

Our proof technique yields another result too: It is PSPACE-complete to check for two given *acyclic* NFAs $\mathcal{A}_1$, $\mathcal{A}_2$ (both accepting the empty word), whether the trees that result from the prefix orders on $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$, respectively, are isomorphic. Note that these two trees are clearly finite (since the automata are acyclic), but the size of $L(\mathcal{A}_i)$ can be exponential in the number of states of $\mathcal{A}_i$. In this sense, acyclic NFAs can be seen as a succinct representation of finite trees. The PSPACE-upper bound for acyclic NFAs follows easily from Lindell's result [19] that isomorphism of explicitly given trees can be checked in logarithmic space.

The second part of this paper studies the isomorphism problem for *regular words*, which were introduced in [6]. A *generalized word* over a finite alphabet Σ is a countable linear order together with a Σ-coloring of the elements. A generalized word is regular if it can be obtained as the least solution (in a certain sense made precise in [6]) of a system $X_1 = t_1, \ldots, X_n = t_n$. Here, every t_i is a finite word over the alphabet $\Sigma \cup \{X_1, \ldots, X_n\}$. For instance, the system $X = abX$ defines the regular word $(ab)^\omega$.

Courcelle [6] gave an alternative characterization of regular words: A generalized word is regular if and only if it is equal to the frontier word of a finitely-branching ordered regular tree, where the leaves are colored by symbols from Σ. Here, the frontier word is obtained by ordering the leaves in the usual left-to-right order (note that the tree is ordered). Alternatively, a regular word can be represented by a DFA $\mathcal{A}$, where the set of final states is partitioned into sets F_a ($a \in \Sigma$); we call such a DFA a *partitioned DFA*. The corresponding regular word is obtained by ordering the language of $\mathcal{A}$ lexicographically and coloring a word $w \in L(\mathcal{A})$ with a if w leads from the initial state to a state from F_a.

A third characterization of regular words was provided by Heilbrunner [12]: A generalized word is regular if it can be obtained from singleton words (i.e., symbols from Σ) using the operations of concatenation, ω-power, $\overline{\omega}$-power and dense shuffle. For a generalized word u, its ω-power (resp. $\overline{\omega}$-power) is the generalized word $uuu\cdots$

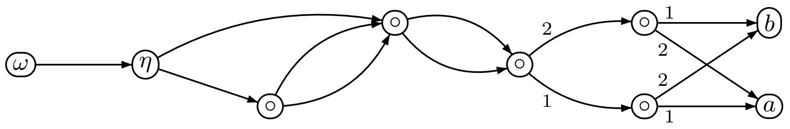

Fig. 1. A dag for the regular word $([abbaabba, abbaabbaabbaabba]^{\eta})^{\omega}$. Nodes labeled with $\circ$ compute the concatenation of their successor nodes. In case the order of the successor nodes matters, we specify it by edge labels.

(resp. $\cdots uuu$). Moreover, the shuffle of generalized words $u_1, \ldots, u_n$ is obtained by choosing a dense coloring of the rationals with colors $\{1, \ldots, n\}$ (up to isomorphism, there is only a single such coloring [21]) and then replacing every i-colored rational by u_i. In fact, Heilbrunner presents an algorithm which computes from a given system of equations (or, alternatively, a partitioned DFA) an expression over the above set of operations (called a *regular expression* in the following) which defines the least solution of the system of equations. A simple analysis of Heilbrunner's algorithm shows that the computed regular expression in general has exponential size with respect to the input system of equations and it is easy to see that this cannot be avoided (take for instance the system $X_i = X_{i+1}X_{i+1}$ ($1 \leq i \leq n$), $X_n = a$, which defines the finite word a^{2^n}).

The next step was taken by Thomas in [24], where he proved that the isomorphism problem for regular words is decidable. For his proof, he uses the decidability of the monadic second-order theory of linear orders; hence his proof does not yield an elementary upper bound for the isomorphism problem for regular words. Such an algorithm was later presented by Bloom and Ésik in [2], where the authors present a polynomial time algorithm for checking whether two given regular expressions define isomorphic regular words. Together with Heilbrunner's algorithm, this yields an exponential time algorithm for checking whether the least solutions of two given systems of equations (or, alternatively, the regular words defined by two partitioned DFAs) are isomorphic. It was asked in [2], whether a polynomial time algorithm for this problem exists.

Our second main result answers this question affirmatively. In fact, we prove that the problem, whether two given partitioned DFAs define isomorphic regular words, is P-complete (Cor. 4.2 and Thm. 4.4). A large part of the long version [18] of this paper deals with the polynomial time upper bound. The first step is simple. By reanalyzing Heilbrunner's algorithm, it is easily seen that from a given partitioned DFA (defining a regular word u) one can compute in *polynomial time* a *succinct representation* of a regular expression for u. This succinct representation consists of a dag (directed acyclic graph), whose unfolding is a regular expression for u. Figure 1 shows an example of such a dag. The second and main step of our proof shows that the polynomial time algorithm of Bloom and Ésik for regular expressions can be refined in such a way that it works (in polynomial time) for succinct regular expressions too. The main tool in our proof is (besides the machinery from [2]) algorithms on compressed strings (see [22] for a survey), in particular Plandowski's result that equality of strings that are represented by *straight-line programs* (i.e., context free grammars that only generate a single word) can be checked in polynomial time [20]. It is a simple observation that an *acyclic* partitioned DFA is basically a straight-line program. Hence, we show how to extend Plandowski's polynomial time algorithm from acyclic partitioned DFAs to general partitioned DFAs.

An immediate corollary of our result is that it can be checked in polynomial time whether the lexicographic orderings on the languages defined by two given DFAs (so called regular linear orderings) are isomorphic. For the special case that the two input DFAs accept well-ordered languages, this was shown in [8]. Let us mention that it is highly undecidable (Σ^1_1-complete) to check, whether the lexicographic orderings on the languages defined by two given deterministic pushdown automata (these are the algebraic linear orderings [3]) are isomorphic [16].

2 Preliminaries

We assume standard notions from automata theory. Let us take a finite alphabet Σ. For $u, v \in \Sigma^*$, we write $u \leq_{\mathsf{pref}} v$ if there exists $w \in \Sigma^*$ with $v = uw$, i.e., u is a *prefix* of v. A language $L \subseteq \Sigma^*$ is *prefix-closed* if $u \leq_{\mathsf{pref}} v \in L$ implies $u \in L$. For a fixed linear order $\leq$ on the alphabet Σ we define the *lexicographic order* $\leq_{\mathsf{lex}}$ on Σ^* as follows: $u \leq_{\mathsf{lex}} v$ if $u \leq_{\mathsf{pref}} v$ or there exist $w, x, y \in \Sigma^*$ and $a, b \in \Sigma$ such that $a < b$, $u = wax$, and $v = wby$. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton (NFA) where Q is the set of states, Σ is the input alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. Then, $\mathcal{A}$ is called *prefix-closed* if $Q = F$ (thus, $L(\mathcal{A})$ is prefix-closed). For a deterministic finite automaton (DFA), δ is a partial map from $Q \times \Sigma$ to Q. A *partitioned DFA* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, (F_a)_{a \in \Gamma})$, where Γ is a finite alphabet, $\mathcal{B} := (Q, \Sigma, \delta, q_0, \bigcup_{a \in \Gamma} F_a)$ is an ordinary DFA and $F_a \cap F_b = \emptyset$ for $a \neq b$. Since $\mathcal{B}$ is a DFA, it follows that the language $L(\mathcal{B})$ is partitioned by the languages $L(Q, \Sigma, \delta, q_0, F_a)$ ($a \in \Gamma$).

We assume that the reader has some basic background in complexity theory, in particular concerning the complexity classes P, PSPACE, and $\mathsf{EXPTIME}$. All completeness results in this paper refer to logspace reductions.

2.1 Trees

A *tree* is a partial order $T = (A; \leq)$, where $\leq$ has a smallest element (the root of the tree; in particular $A \neq \emptyset$) and for every $a \in A$, the set $\{b \in A \mid b \leq a\}$ is finite and linearly ordered by $\leq$. We write $a \lessdot b$ if $a < b$ and there does not exist $c \in A$ with $a < c < b$. Then $(A; \lessdot)$ is a tree in the graph theoretical sense (sometimes, it is also called a successor tree). For two trees T_1 and T_2, we write $T_1 \cong T_2$ in case T_1 and T_2 are isomorphic. A *tree over the finite alphabet* Σ is a pair $T = (L; \leq_{\mathsf{pref}})$, where $L \subseteq \Sigma^*$ is a language with $\varepsilon \in L$. Note that T is indeed a tree in the above sense (ε is the root). If L is prefix-closed, then, clearly, T is a finitely branching tree.

A countable tree T is called *regular* if T has only finitely many subtrees up to isomorphism, see e.g. [4, 24]. Equivalently, a countable tree is regular if it is isomorphic to a tree of the form $(L; \leq_{\mathsf{pref}})$, where L is a regular language with $\varepsilon \in L$. If L is accepted by the DFA $\mathcal{A}$ and all final states can be reached from the initial state, then the subtrees of $(L; \leq_{\mathsf{pref}})$ correspond to the final states of $\mathcal{A}$. Note that by our definition, a regular tree needs not be finitely branching. A finitely branching tree is regular if and only if it is the unfolding of a finite directed graph [4].

Our first definition of a regular tree (having only finitely many subtrees up to isomorphism) makes sense for other types of trees as well, e.g. for node-labeled trees or ordered trees (where the children of a node are linearly ordered). These variants of regular trees can be generated by finite automata as well. For instance, a node-labeled regular tree $(L; \leq_{\mathsf{pref}}, (L_a)_{a \in \Gamma})$, where Γ is a finite labeling alphabet and L_a is the set of a-labeled nodes can be specified by a partitioned DFA $(Q, \Sigma, \delta, q_0, (F_a)_{a \in \Gamma})$ with $L_a = L(Q, \Sigma, \delta, q_0, F_a)$ and $L = \bigcup_{a \in \Gamma} L_a$. We do not consider node labels in this paper, since it makes no difference for the isomorphism problem (node labels can be eliminated by adding additional children to nodes). Ordered regular trees are briefly discussed in the long version [18] of this paper.

2.2 Linear orders and generalized words

See [21] for a thorough introduction into linear orders. Let η be the order type of the rational numbers, ω be the order type of the natural numbers, and $\overline{\omega}$ be the order type of the negative integers. With $\mathbf{n}$ we denote a finite linear order with n elements. Let $\Lambda = (L; \leq)$ be a linear order. An *interval* of Λ is a subset $I \subseteq L$ such that $x < z < y$ and $x, y \in I$ implies $z \in I$. The predecessor (resp., successor) of $x \in L$ is a largest (resp., smallest) element of $\{y \in L \mid y < x\}$ (resp., $\{y \in L \mid x < y\}$). Of course, the *predecessor* (resp., *successor*) of x need not exist, but if it exists then it is unique. The linear order Λ is *dense* if L consists of at least two elements, and for all $x < y$ there exists z with $x < z < y$. By Cantor's theorem, every countable dense linear order, which neither has a smallest nor largest element is isomorphic to η. Hence, if we take symbols 0 and 1 with $0 < 1$, then $(\{0,1\}^*1; \leq_{\mathsf{lex}}) \cong \eta$. The linear order Λ is *scattered* if there does not exist an injective order morphism $\varphi : \eta \to \Lambda$. Clearly, ω, $\overline{\omega}$, as well as every finite linear order are scattered. A linear order is *regular* if it is isomorphic to a linear order $(L; \leq_{\mathsf{lex}})$ for a regular language L. For instance, ω, $\overline{\omega}$, η, and every finite linear order are regular linear orders.

Generalized words are countable colored linear orders. Let Σ be a finite alphabet. A *generalized word* (or simply word) u over Σ is a triple $(L; \leq, \tau)$ such that L is a finite or countably infinite set, $\leq$ is a linear order on L and $\tau : L \to \Sigma$ is a coloring of L. The alphabet $\mathsf{alph}(u)$ equals the image of τ. If L is finite, we obtain a finite word in the usual sense. Moreover, $u = (L; \leq, \tau)$ is scattered if $(L; \leq)$ is scattered. We write $u \cong v$ for generalized words u and v, if u and v are isomorphic.

There is a natural operation of concatenation of two generalized words. Let $u_1 = (L_1; \leq_1, \tau_1)$ and $u_2 = (L_2; \leq_2, \tau_2)$ be generalized words with $L_1 \cap L_2 = \emptyset$. Then $u_1 u_2$ is the generalized word $(L_1 \cup L_2; \leq, \tau_1 \cup \tau_2)$, where $x \leq y$ if and only if either $x, y \in L_1$ and $x \leq_1 y$, or $x, y \in L_2$ and $x \leq_2 y$, or $x \in L_1$ and $y \in L_2$. Similarly, we can define the ω-power (resp., $\overline{\omega}$-power) of a generalized word u as the generalized word that results from ω (resp. $\overline{\omega}$) by replacing every point by a copy of u. So, intuitively, $u^\omega = uuu\cdots$ and $u^{\overline{\omega}} = \cdots uuu$. Finally, we need the shuffle operator. Given generalized words $u_1, \ldots, u_n$, we let $[u_1, \ldots, u_n]^\eta$ be the generalized word that is obtained from η as follows: Take a coloring of η with colors $1, \ldots, n$ such that for all $x, y \in \mathbb{Q}$ with $x < y$ and all $1 \leq i \leq n$, there exists $x < z < y$ such that z has color i (it can be shown that up to isomorphism there is a unique such dense coloring [21]). Then the *shuffle of* $u_1, \ldots, u_n$, denoted by $[u_1, \ldots, u_n]^\eta$, is obtained by replacing every i-colored rational by a copy of the generalized word u_i. Since $[u_1, \ldots, u_n]^\eta$ is invariant under permutations of the u_i, we also sometimes use the notation X^η for a finite set X of generalized words. The least set of words which contains the singleton words a for $a \in \Sigma$ and is closed under concatenation, ω-power, $\overline{\omega}$-power, and shuffle is called the set of *regular words* over Σ, denoted $\mathrm{Reg}(\Sigma)$. Note that this definition implies that every regular word is non-empty, i.e., its domain is a non-empty set. Clearly, every regular word can be described by a *regular expression* over the above operations, but this regular expression is in general not unique. Given a regular expression e, we define the corresponding regular word by $\mathsf{val}(e)$.

By a result of Heilbrunner [12], regular words can be characterized by partitioned DFAs as follows: Let $\mathcal{A} = (Q, \Gamma, \delta, q_0, (F_a)_{a \in \Sigma})$ be a partitioned DFA, and let $\mathcal{B} = (Q, \Gamma, \delta, q_0, \bigcup_{a \in \Sigma} F_a)$. Let us fix a linear order on the alphabet Γ, so that the lexicographic order $\leq_{\mathsf{lex}}$ is defined on Γ^*. Then we denote with $w(\mathcal{A})$ the generalized word $w(\mathcal{A}) = (L(\mathcal{B}); \leq_{\mathsf{lex}}, \tau)$, where $\tau(u) = a$ ($a \in \Sigma$, $u \in L(\mathcal{B})$) if and only if $u \in L(Q, \Gamma, \delta, q_0, F_a)$. It is easy to construct from a given regular expression e a partitioned DFA $\mathcal{A}$ with $\mathsf{val}(e) \cong w(\mathcal{A})$, see e.g. [24, proof of Prop. 2] for a simple construction. The other direction is more difficult. Heilbrunner has shown in [12] how to compute from a given partitioned DFA $\mathcal{A}$ (such that $w(\mathcal{A})$ is non-empty) a regular expression e with $\mathsf{val}(e) \cong w(\mathcal{A})$.[1] Unfortunately, the size of the regular expression produced by Heilbrunner's algorithm is exponential in the size of $\mathcal{A}$. On the other hand, reanalyzing Heilbrunner's algorithm shows that a succinct representation of a regular expression for $w(\mathcal{A})$ can be produced in polynomial time. This succinct representation is a dag (directed acyclic graph), where multiple occurrences of the same regular subexpression are represented only once. We denote such dags with $\mathbb{A}, \mathbb{B}$, etc. The regular word represented by the dag $\mathbb{A}$ is again denoted by $\mathsf{val}(\mathbb{A})$.

3 Isomorphism problem for regular trees

In this section, we investigate the isomorphism problem for (unordered) regular trees. We consider two input representations for regular trees: DFAs and NFAs. It turns out that while the isomorphism problem for DFA-represented regular trees is P-complete, the same problem becomes EXPTIME-complete for NFA-represented regular trees. Moreover, we show that for *finite* trees that are succinctly represented by *acyclic* NFAs, isomorphism is PSPACE-complete.

Let us start with upper bounds. Our proof of the following theorem is based on an algorithm similar to the partition refinement algorithm for checking bisimilarity of finite state systems [14]. The statement for NFAs clearly follows from the statement for DFAs using the powerset construction for transforming NFAs into DFAs.

Theorem 3.1. *For two given DFAs (resp., NFAs) $\mathcal{A}_1, \mathcal{A}_2$ such that $\varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ one can decide in polynomial time (resp., exponential time) whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

[1] In fact, Heilbrunner [12] speaks about systems of equations and their least solutions instead of partitioned DFAs. These two formalisms can be efficiently transformed into each other.

For acyclic NFAs, we can improve the upper bound from Thm. 3.1 to PSPACE.

Theorem 3.2. *For two given acyclic NFAs $\mathcal{A}_1$, $\mathcal{A}_2$ such that $\varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ one can decide in polynomial space whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

The proof of Thm. 3.2 is based on two facts: (i) Given an acyclic NFA $\mathcal{A}$ we can compute an explicit representation of the finite tree $(L(\mathcal{A}); \leq_{\mathsf{pref}})$ (using e.g. adjacency lists) by a transducer, whose working tape is polynomially bounded, and (ii) isomorphism for explicitly given finite trees can be checked in logspace [19].

Concerning lower bounds, our main result is:

Theorem 3.3. *It is* EXPTIME*-hard (and hence* EXPTIME*-complete) to decide for two given prefix-closed NFAs $\mathcal{A}_1$, $\mathcal{A}_2$, whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

It is straightforward to prove PSPACE-hardness of the problem in Thm. 3.3. If Σ is the underlying alphabet of a given NFA $\mathcal{A}$, then $(L(\mathcal{A}); \leq_{\mathsf{pref}})$ is a full $|\Sigma|$-ary tree if and only if $L(\mathcal{A}) = \Sigma^*$. But universality for NFAs is PSPACE-complete [23]. The proof for the EXPTIME lower bound stated in Thm. 3.3 is more involved. Here is a rough outline: EXPTIME coincides with alternating polynomial space [5]. Checking whether a given input is accepted by a polynomial space bounded alternating Turing machine M amounts to evaluate a Boolean expression whose gates correspond to configurations of M. Using a construction from [13], the evaluation problem for (finite) Boolean expressions can be reduced to the isomorphism problem for (finite) trees. In our case, the Boolean expression will be infinite. Nevertheless, the infinite Boolean expressions we have to deal with can be evaluated because on every infinite path that starts in the root (the output gate) there is either an and-gate, where one of the inputs is a false-gate, or an or-gate, where one of the inputs is a true-gate. Applying the construction from [13] to an infinite Boolean expression (that arises from our construction) yields two infinite trees, which are isomorphic if and only if our infinite Boolean expression evaluates to true. Luckily, these two trees turn out to be regular, and they can be represented by small NFAs. Using a similar construction, but starting with an alternating polynomial time machine (instead of an alternating polynomial space machine), we can prove:

Theorem 3.4. *It is* PSPACE*-hard (and hence* PSPACE*-complete) to decide for two given prefix-closed acyclic NFAs $\mathcal{A}_1$, $\mathcal{A}_2$, whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

Finally, by a reduction from the P-complete monotone circuit value problem [11] (which uses again the reduction from the evaluation problem for Boolean expressions to the isomorphism problem for explicitly given finite trees [13]), we get the next result.

Theorem 3.5. *It is* P*-hard (and hence* P*-complete) to decide for two given prefix-closed acyclic DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, whether $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$.*

4 Isomorphism problem for regular words

In this section we study the isomorphism problem for regular words that are represented by partitioned DFAs. We prove that this problem as well as the isomorphism problem for regular linear orders that are represented by DFAs are P-complete. It follows that the isomorphism problem for regular linear orders that are represented by NFAs can be solved in exponential time. We show that this problem is also PSPACE-hard.

4.1 Upper bounds

In Section 2.2 we mentioned that Heilbrunner's algorithm [12] transforms a given partitioned DFA $\mathcal{A}$ into a succinct regular expression $\mathbb{A}$ (in form of a dag) for the regular word $w(\mathcal{A})$. This motivates the following result:

Theorem 4.1. *For two given dags $\mathbb{A}_1$ and $\mathbb{A}_2$ one can decide in polynomial time, whether $\mathsf{val}(\mathbb{A}_1) \cong \mathsf{val}(\mathbb{A}_2)$.*

The next result is an immediate corollary of Thm. 4.1 and [12].

Corollary 4.2. *For two given partitioned DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ one can decide in polynomial time whether $w(\mathcal{A}_1) \cong w(\mathcal{A}_2)$.*

Our proof of Thm. 4.1 is quite long and technical. But essentially, we use the same strategy as in [2]. Recall that Bloom and Ésik prove in [2] that for two given (non-succinct) regular expressions e_1, e_2 it can be decided in polynomial time, whether they represent the same regular word. Let us briefly explain their strategy.

A central concept in [2] is the notion of a block of a generalized word. Blocks allow to condensate a generalized word to a coarser word (whose elements are the blocks of the original word). Let $u = (L; \leq, \tau)$ be a generalized word over the alphabet Σ. A *subword* of u is an interval I of the linear order $(L; \leq)$) together with the coloring τ restricted to I. A *uniform subword* of u is a subword that is isomorphic to a shuffle Γ^η for some $\Gamma \subseteq \Sigma$. A uniform subword is a *maximal uniform subword* if it is not properly contained in another uniform subword. Now let v be a subword such that no point of v is contained in a uniform subword of u. Then v is *successor-closed* if for each point p of v, whenever the successor and the predecessor of p exist, they are contained in v as well. A successor-closed subword is minimal if it does not strictly contain another successor-closed subword. Finally, a *block* of the generalized word u is either a maximal uniform subword of u or a minimal successor-closed subword of u. A regular word which consists of a single block is called *primitive.*[2] By [2] a generalized word u is primitive if and only if it is of one of the following forms (where $x, z \in \Sigma^+$, $y \in \Sigma^*$): A finite non-empty word, a scattered word of the form $x^{\overline{\omega}}y$, a scattered word of the form yz^ω, a scattered word of the form $x^{\overline{\omega}}yz^\omega$, or a uniform word ($\Gamma^\eta$ for some $\Gamma \subseteq \Sigma$). Let $D(\Sigma)$ be the set of all primitive words over Σ.

Let u be a regular word. Each point p of u belongs to some unique block $\mathrm{Bl}(p)$, which induces a regular (and hence primitive) word. Moreover we can order the blocks of u linearly by setting $\mathrm{Bl}(p) < \mathrm{Bl}(q)$ if and only if $p < q$. The order obtained that way is denoted $(\mathrm{Bl}(u); \leq)$. Then we extend the order $(\mathrm{Bl}(u); \leq)$ to a generalized word $\widehat{u}$ over the alphabet $D(\Sigma)$, called the *skeleton* of u, by labeling each block with the corresponding isomorphic word in $D(\Sigma)$. Implicitly, it is shown in [2] that for every regular word u there exists a *finite* subset of $D(\Sigma)$ such that every block of u is isomorphic to a generalized word from that finite subset. Moreover, $\widehat{u}$ is a regular word over that finite subset of $D(\Sigma)$. Bloom and Ésik have shown that two regular words are isomorphic if and only if their skeletons are isomorphic [2, Cor. 73].

[2] In combinatorics on words, a finite word is called primitive, if it is not a proper power of a non-empty word. Our notion of a primitive word should not be confused with this definition.

From two given regular expression e_1 and e_2, Bloom and Ésik compute in polynomial time two "simpler" expressions f_1 and f_2 (over a finite alphabet, which consists of a finite subset of $D(\Sigma)$) such that $\mathsf{val}(f_i)$ is the skeleton of $\mathsf{val}(e_i)$. Here, "simpler" means that the height of the expression f_i is strictly smaller than the height of e_i. The above mentioned Cor. 73 from [2] allows to replace e_1, e_2 by f_1, f_2. This step is iterated until one of the two expressions denotes a primitive regular word. If at that point the other expression does not denote a primitive word, then the two initial regular words are not isomorphic. On the other hand, if both regular expressions denote primitive regular words, then one faces the problem of checking whether two given primitive regular words are isomorphic. It is straightforward to do this in polynomial time.

For succinct expressions (i.e., dags), we use the same strategy. Given two dags $\mathbb{A}_1$ and $\mathbb{A}_2$, we compute in polynomial time new dags $\mathbb{B}_1$ and $\mathbb{B}_2$ such that (i) $\mathsf{val}(\mathbb{B}_i)$ is the skeleton of $\mathsf{val}(\mathbb{A}_i)$ and (ii) the height of $\mathbb{B}_i$ is strictly smaller than the height of $\mathbb{A}_i$. Note that the notion of "height" makes sense for dags as well. It is the maximal length of a path in the dag. To obtain a polynomial time algorithm at the end, several problems have to be addressed. First of all, the transformation of a dag $\mathbb{A}$ into a dag $\mathbb{B}$ such that $\mathsf{val}(\mathbb{B})$ is the skeleton of $\mathsf{val}(\mathbb{A})$ must be accomplished in polynomial time. But even if we can achieve this, an overall polynomial running time is not guaranteed, since we have to iterate this transformation. If for instance, the size of $\mathbb{B}$ (let us define the size of a dag as the number of edges) would be twice the size of $\mathbb{A}$, then this would result into an overall exponential blow-up. But fortunately, our transformation of $\mathbb{A}$ into $\mathbb{B}$ only involves an additive blow-up, which is polynomial at each iteration. Finally, at the end, we have to check isomorphism for two primitive regular words that are succinctly represented by dags. It is not obvious to do this in polynomial time. In fact, our algorithm for solving this problem makes essential use of known results for compressed words.

Let us explain this in more detail. A dag, where only the alphabet symbols and the operation of concatenation is used (no ω- and $\overline{\omega}$-powers and no shuffles) is also known as a *straight-line program (SLP)*. Alternatively, it can be seen as an acyclic context-free grammar, where each nonterminal is the left-hand side of a unique production. Such a context-free grammar generates a single finite word. Moreover, the length of this word can be exponential in the size of the SLP. Hence the SLP can be seen as a compressed representation of the finite word. In recent years, a lot of effort was spent on the development of efficient algorithms for SLP-represented finite words: Our polynomial time algorithm for primitive regular words that are given by dags uses a seminal result from this area: It can be checked in polynomial time, whether the word represented by a first SLP is a factor of the word represented by a second SLP (compressed pattern matching). The best algorithm for compressed pattern matching has a cubic running time [17]. Note that as a corollary, it can be checked in polynomial time, whether two given SLPs represent the same finite word. This result was first shown by Plandowski [20].

4.2 Lower bounds for regular linear orders

Let us now turn to lower bounds for the isomorphism problem for regular words. In fact, all these lower bounds already hold for a unary alphabet, i.e., they hold for regular linear orders. The results in this section nicely contrast the results from Section 3, where we studied the isomorphism problem for the prefix order trees on regular languages. In this section, we replace the prefix order by the lexicographical order.

Theorem 4.3. *For every finite alphabet Σ, it is* P*-hard (and hence* P*-complete) to decide for two given dags $\mathbb{A}_1$ and $\mathbb{A}_2$ over the alphabet Σ, whether* $\mathsf{val}(\mathbb{A}_1) \cong \mathsf{val}(\mathbb{A}_2)$.

As for the proof of Thm. 3.5, the proof of Thm. 4.3 is based on a reduction from the monotone circuit value problem. We do not know, whether the lower bound from Thm. 4.3 holds for ordinary expressions (instead of dags) too.

Theorem 4.4. *It is* P*-hard (and hence* P*-complete) to decide for two given DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, whether $(L(\mathcal{A}_1); \leq_{\mathsf{lex}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{lex}})$.*

Proof. Note that by Cor. 4.2 the problem belongs to P. For P-hardness, it suffices by Thm. 4.3 to construct in logspace from a given dag $\mathbb{A}$ (over a unary terminal alphabet) a DFA $\mathcal{A}$ such that the linear order $\mathsf{val}(\mathbb{A})$ is isomorphic to $(L(\mathcal{A}); \leq_{\mathsf{lex}})$. But this is accomplished by the construction in the proof of [24, Prop. 2]. □

Cor. 4.2 implies that it can be checked in $\mathsf{EXPTIME}$ whether the lexicographical orderings on two regular languages, given by NFAs, are isomorphic. We do not know whether this upper bound is sharp. Currently, we can only prove a lower bound of PSPACE:

Theorem 4.5. *It is* PSPACE*-hard to decide for two given NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, whether $(L(\mathcal{A}_1); \leq_{\mathsf{lex}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{lex}})$.*

Proof. We prove PSPACE-hardness by a reduction from the PSPACE-complete problem whether a given NFA $\mathcal{A}$ over the terminal alphabet $\{a,b\}$ accepts $\{a,b\}^*$ [23]. So let $\mathcal{A}$ be an NFA over the terminal alphabet $\{a,b\}$ and let $K = L(\mathcal{A})$. Let $\Sigma = \{0,1,a,b,\$_1,\$_2\}$ and fix the following order on Σ: $\$_1 < 0 < 1 < \$_2 < a < b$. Under this order, $(\{0,1\}^*1; \leq_{\mathsf{lex}}) \cong (\{a,b\}^*b; \leq_{\mathsf{lex}}) \cong \eta$.

It is straightforward to construct from $\mathcal{A}$ in logspace an NFA for the language

$$L \;=\; (\{a,b\}^* b\, \$_1) \;\cup\; (K\, b\, \{0,1\}^* 1) \;\cup\; (\{a,b\}^* b\, \$_2). \tag{1}$$

It follows that

$$(L; \leq_{\mathsf{lex}}) \cong \sum_{w \in \{a,b\}^* b} \mathcal{L}(w) \quad \text{with} \quad \mathcal{L}(w) \cong \begin{cases} \mathbf{1} + \eta + \mathbf{1} & \text{if } w \in K \\ \mathbf{2} & \text{else.} \end{cases}$$

(the sum is taken over all words from $\{a,b\}^*b$ in lexicographic order). If $K \neq \{a,b\}^*$, then $(L; \leq_{\mathsf{lex}})$ contains an interval isomorphic to $\mathbf{2}$. Hence $(L; \leq_{\mathsf{lex}}) \ncong \eta$. On the other hand, if $K = \{a,b\}^*$, then $(L; \leq_{\mathsf{lex}}) \cong (\mathbf{1} + \eta + \mathbf{1}) \cdot \eta \cong \eta$. This proves the theorem. □

The proof of Thm. 4.5 shows that it is PSPACE-hard to check for a given NFA $\mathcal{A}$, whether $(L(\mathcal{A}); \leq_{\mathsf{lex}}) \cong \eta$. In fact, this problem turns out to be PSPACE-complete, see the long version [18] for details.

In [9] it is shown that the problem, whether $(L; \leq_{\mathsf{lex}}) \cong \eta$ for a given context-free language, is undecidable. This result is shown by a reduction from Post's correspondence problem. Note that this result can be also easily deduced using the technique

	DFA	NFA
acyclic	P-complete	PSPACE-complete
arbitrary		$\mathsf{EXPTIME}$-complete

Table 1. Main results for the isomorphism problem for regular trees

	DFA	NFA
acyclic	$\mathsf{C_=L}$-complete	$\mathsf{C_=P}$-complete
arbitrary	P-complete	PSPACE-hard, in $\mathsf{EXPTIME}$

Table 2. Main results for the isomorphism problem for regular linear orders

from the above proof: If we start with a pushdown automaton for $\mathcal{A}$ instead of an NFA, then the language L from (1) is context-free. Hence, $(L; \leq_{\mathsf{lex}}) \cong \eta$ if and only if $L(\mathcal{A}) = \{a,b\}^*$. The latter property is a well-known undecidable problem.

In Section 3 we also studied the isomorphism problem for finite trees that are succinctly given by the prefix order on the finite language accepted by an acyclic DFA (resp., NFA). To complete the picture, we should also consider the isomorphism problem for linear orders that consist of a lexicographically ordered finite language, where the latter is represented by an acyclic DFA (resp., NFA). Of course, this problem is somehow trivial, since two finite linear orders are isomorphic if and only if they have the same cardinality. Hence, we have to consider the problem whether two given acyclic DFAs (resp. NFAs) accept languages of the same cardinality. The complexity of these problems is analyzed in the long version [18]. Straightforward arguments show that checking whether two acyclic DFAs (resp. NFAs) accept languages of the same cardinality is complete for the counting class $\mathsf{C_=L}$ (resp., $\mathsf{C_=P}$), see [18] for definitions.

5 Conclusion and open problems

Table 1 (Table 2) summarizes our complexity results for the isomorphism problem for regular trees (regular linear orders). Let us conclude with some open problems. As can be seen from Table 2, there is a complexity gap for the isomorphism problem for regular linear orders that are represented by NFAs. This problem belongs to $\mathsf{EXPTIME}$ and is PSPACE-hard. Another interesting problem concerns the equivalence problem for straight-line programs (i.e., dags that generate finite words, or equivalently, acyclic partitioned DFAs, or equivalently, context-free grammars that generate a single word). Plandowski has shown that this problem can be solved in polynomial time. Recall that this result is fundamental for our polynomial time algorithm for dags (Thm. 4.1). In [10], it was conjectured that the equivalence problem for straight-line programs is P-complete, but this is still open.

References

1. V. Bárány, E. Grädel, and S.Rubin. Automata-based presentations of infinite structures. In *Finite and Algorithmic Model Theory*, number 379 in London Mathematical Society Lecture Notes Series. Cambridge University Press, 2011. to appear.
2. S. L. Bloom and Z. Ésik. The equational theory of regular words. *Inf. Comput.*, 197(1-2):55–89, 2005.
3. S. L. Bloom and Z. Ésik. Algebraic linear orderings. Technical report, arXiv.org, 2010. `http://arxiv.org/abs/1002.1624`.
4. D. Caucal. On infinite terms having a decidable monadic theory. In *Proc. MFCS 2002*, LNCS 2420, pages 165–176. Springer, 2002.
5. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
6. B. Courcelle. Frontiers of infinite trees. *ITA*, 12(4), 1978.
7. B. Courcelle. The definability of equational graphs in monadic second-order logic. In *Proc. ICALP'89*, LNCS 372, pages 207–221. Springer, 1989.
8. Z. Ésik. Representing small ordinals by finite automata. In *Proc. DCFS 2010*, volume 31 of *EPTCS*, pages 78–87, 2010.
9. Z. Ésik. An undecidable property of context-free linear orders. *Inf. Process. Lett.*, 111(3):107–109, 2011.
10. L. Gasieniec, A. Gibbons, and W. Rytter. Efficiency of fast parallel pattern searching in highly compressed texts. In *Proc. MFCS'99*, LNCS 1672, pages 48–58. Springer, 1999.
11. L. M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–99, 1977.
12. S. Heilbrunner. An algorithm for the solution of fixed-point equations for infinite words. *ITA*, 14(2):131–141, 1980.
13. B. Jenner, J. Köbler, P. McKenzie, and J. Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003.
14. P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1), 1990.
15. B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: richness and limitations. *Logical Methods in Computer Science*, 3(2):2:2, 18 pp. (electronic), 2007.
16. D. Kuske, J.. Liu, and M. Lohrey. The isomorphism problem on classes of automatic structures with transitive relations. submitted for publication, extended version of a paper presented at LICS 2010, 2011.
17. Y. Lifshits. Processing compressed texts: A tractability border. In *Proc. CPM 2007*, LNCS 4580, pages 228–240. Springer, 2007.
18. M. Lohrey and C. Mathissen Isomorphism of regular trees and words. Technical report, arXiv.org, 2011. `http://arxiv.org/abs/1102.2782`.
19. S. Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proc. STOC'92*, pages 400–404. ACM Press, 1992.
20. W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, LNCS 855, pages 460–470. Springer, 1994.
21. J. Rosenstein. *Linear Ordering*. Academic Press, 1982.
22. W. Rytter. Grammar compression, LZ-encodings, and string algorithms with implicit input. In *Proc. ICALP 2004*, LNCS 3142, pages 15–27. Springer, 2004.
23. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proc. STOCS'73*, pages 1–9. ACM Press, 1973.
24. W. Thomas. On frontiers of regular trees. *ITA*, 20(4):371–381, 1986.