Isomorphism of regular trees and words

Markus Lohrey[a], Christian Mathissen[a]

[a]*Institut für Informatik, Universität Leipzig, Germany*

Abstract

The computational 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. Similar techniques can be used to show that one can check in polynomial time whether a given regular linear order has a non-trivial automorphism. This improves a recent decidability result of Kuske [21].

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 [20] that even for automatic

Email addresses: lohrey@informatik.uni-leipzig.de (Markus Lohrey), mathissen@informatik.uni-leipzig.de (Christian Mathissen)

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 [23], this result was further improved to automatic order trees and automatic linear orders. On the decidability side, Courcelle proved that the isomorphism problem for equational graphs is decidable [8]. 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 spectrum. 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 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 (the empty word is the root of the tree). 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 [18], see Section 3.1. 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, see Section 3.2.2. The proof of the EXPTIME lower bound uses three main ingredients: (i) EXPTIME coincides with alternating polynomial space [6], (ii) a construction from [16], 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 [35].[1]. 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 [25] 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 [7]. A *generalized word* over an 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 [7]) 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 [7] 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*, see also [2] where the term A-automaton is used. 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 [14]: 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 $uuu\cdots$ (resp. $\cdots uuu$). Moreover, the dense 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 [33]) and then replacing every

[1] This construction is used in [35] to prove that the universality problem for NFAs is PSPACE-complete.

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.[2] The next step was taken by Thomas in [36], 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 presented later by Bloom and Ésik in [3], 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 [3], 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. A large part 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. The second and main step of the 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 uses (besides the machinery from [3]) algorithms on compressed strings (see [26, 34] for surveys). In particular, we use the 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; this result was

[2] 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}.

independently shown in [15, 28, 31]. It is a simple observation that an *acyclic* partitioned DFA is basically a straight-line program. Hence, we show how to extend equality checking for 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 [9]. 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 [4]) are isomorphic [23].

In Section 4.7 we finally present a polynomial time algorithm for checking whether a given regular word that is represented by a partitioned DFA has a non-trivial automorphism. This improves a recent decidability result of Kuske [21]. For the proof, we reuse our machinery developed for the isomorphism problem for regular words.

An extended abstract of this paper appeared as [27].

2. Preliminaries

As usual, for a function f and a subset A of the domain of f, $f{\restriction}_A$ denotes the restriction of f to A. Let us take a finite alphabet Σ. The length of a finite word $u \in \Sigma^*$ is denoted by $|u|$. Let $\Sigma^+ = \{u \in \Sigma^* \mid |u| > 0\}$, $\Sigma^k = \{u \in \Sigma^* \mid |u| = k\}$, $\Sigma^{\leq k} = \{u \in \Sigma^* \mid |u| \leq k\}$, and $\Sigma^{\geq k} = \{u \in \Sigma^* \mid |u| \geq k\}$. 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. We write $u <_{\mathsf{pref}} v$ if $u \leq_{\mathsf{pref}} v$ and $u \neq v$. For a language $L \subseteq \Sigma^*$ let $\mathsf{pref}(L) = \{u \in \Sigma^* \mid \exists v \in L : u \leq_{\mathsf{pref}} v\}$. 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 words w, x, y and $a, b \in \Sigma$ such that $a < b$, $u = wax$, and $v = wby$.

2.1. Complexity theory

We assume that the reader has some basic background in complexity theory, in particular concerning the complexity classes NL, P, PSPACE, and $\mathsf{EXPTIME}$, see e.g. [30]. All completeness results in this paper refer to logspace reductions.

A PSPACE-transducer is a deterministic Turing machine with a read-only input tape, a write-only output tape and a work tape, whose length is bounded by $n^{O(1)}$, where n is the input length. The output is written from left to right on the output tape, i.e., in each step the transducer either outputs a new symbol on the output tape, in which case the output head moves one cell to the right, or the transducer does not output a new symbol in which case the output head does not move. Moreover, we assume that the transducer terminates for every input. This implies that a PSPACE-transducer computes a mapping $f : \Sigma^* \to \Theta^*$, where $|f(w)|$ is bounded by $2^{|w|^{O(1)}}$. We need the following simple lemma:

Lemma 1. *Assume that the mapping $f : \Sigma^* \to \Theta^*$ can be computed by a* PSPACE*-transducer and let $L \subseteq \Theta^*$ be a language in* $\mathsf{NSPACE}(\log^k(n))$ *for some constant k. Then $f^{-1}(L)$ belongs to* PSPACE*.*

Proof. The proof uses the same idea that shows that the composition of two logspace computable mappings is again logspace computable. Let $w \in \Sigma^*$ be an input. Basically, we run the $\mathsf{NSPACE}(\log^k(n))$-algorithm for L on the input $f(w)$. But since f can be computed by a PSPACE-transducer (which can generate an exponentially long output) the length of $f(w)$ can be only bounded by $2^{|w|^{O(1)}}$. Hence, we cannot construct $f(w)$ explicitly. But this is not necessary. We only store a pointer to some position $f(w)$ (this pointer needs space $|w|^{O(1)}$) while running the $\mathsf{NSPACE}(\log^k(n))$-algorithm for L. Each time, this algorithm needs the i^{th} letter of $f(w)$, we run the PSPACE-transducer for L until the i^{th} output symbol is generated. The first $i-1$ symbols of $f(w)$ are not written on the output tape. Note that the $\mathsf{NSPACE}(\log^k(n))$-algorithm for L needs space $\log^k(2^{|w|^{O(1)}}) = |w|^{O(1)}$ while running on $f(w)$. Hence, the total space requirement is bounded by $|w|^{O(1)}$. □

An *alternating Turing machine* is an ordinary nondeterministic Turing machine, where in addition the set of states Q is partitioned into existential states ($Q_\exists$) and universal states ($Q_\forall$). A configuration, where the current state is existential (resp., universal) is called an existential (resp., universal) configuration. Let us assume that M is an alternating Turing machine without infinite computation paths. Then, we define inductively the notion of an *accepting configuration* as follows: If c is an existential configuration, then c is accepting if and only if c has an accepting successor configuration. If c

is a universal configuration, then c is accepting if and only if all successor configurations of c are accepting. Note that a universal configuration without successor configurations is accepting, whereas an existential configuration without successor configurations is not accepting. An input x is accepted by M (briefly, $x \in L(M)$) if and only if the initial configuration with input x is accepting.

The complexity class $\mathsf{C}_=\mathsf{P}$ consists of all languages $L \subseteq \Sigma^*$ such that there exist nondeterministic polynomial time Turing machines M_1 and M_2 with input alphabet Σ such that for every input $w \in \Sigma^*$: $w \in L$ if and only if the number of accepting computations of M_1 on input w equals the number of accepting computations of M_2 on input w. If we replace in this definition nondeterministic polynomial time Turing machines by nondeterministic logspace Turing machines, we obtain the class $\mathsf{C}_=\mathsf{L}$.

2.2. Finite automata and transducer

Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton, briefly *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. A state $q \in Q$ is *accessible* (resp. *coaccessible*), if q can be reached from the initial state q_0 (resp., if a final state from F can be reached from q). We say that $\mathcal{A}$ is accessible (resp., coaccessible), if every state of $\mathcal{A}$ is accessible (resp, coaccessible). An NFA $\mathcal{A}$ is called *prefix-closed* if every state of $\mathcal{A}$ is a final state. In that case, the language $L(\mathcal{A})$ is prefix-closed, i.e., $L(\mathcal{A}) = \mathsf{pref}(L(\mathcal{A}))$. Moreover, if $\mathcal{A}$ is coaccessible and the prefix-closed NFA $\mathcal{B}$ results from $\mathcal{A}$ by making every state final, then clearly $L(\mathcal{B}) = \mathsf{pref}(L(\mathcal{A}))$. For a DFA (deterministic finite automaton), δ is a partial map from $Q \times \Sigma$ to Q. Sometimes, we will also deal with NFAs (DFAs) without an initial state. If $\mathcal{A}$ is an NFA without an initial state and q is a state of $\mathcal{A}$, then $L(\mathcal{A}, q)$ is the language accepted by $\mathcal{A}$, when q is declared to be the initial state. We will need the following simple lemma, which is probably folklore:

Lemma 2. *For a given a DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, we can compute the cardinality $|L(\mathcal{A})| \in \mathbb{N} \cup \{\infty\}$ in polynomial time.*

Proof. W.l.o.g we can assume that $\mathcal{A}$ is accessible and coaccessible. Then $L(\mathcal{A})$ is finite if and only if $\mathcal{A}$ is acyclic. So assume that $\mathcal{A}$ is acyclic. Since $\mathcal{A}$ is deterministic, the size of $L(\mathcal{A})$ equals the number of paths from q_0 to F. Now, in a directed acyclic graph, the number of paths from a source node to all other nodes can be easily computed by dynamic programming in polynomial time. □

A *partitioned DFA* [2] is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, (F_a)_{a\in\Gamma})$, where Γ is a finite alphabet, $F_a \subseteq Q$ for all $a \in \Gamma$, $F_a \cap F_b = \emptyset$ for $a \neq b$, and $\mathcal{B} = (Q, \Sigma, \delta, q_0, \bigcup_{a\in\Gamma} F_a)$ is an ordinary DFA. Since $\mathcal{B}$ is a DFA, it follows that the language $L(\mathcal{B})$ is partitioned by the languages $L(\mathcal{A}_a)$, where $\mathcal{A}_a = (Q, \Sigma, \delta, q_0, F_a)$ $(a \in \Gamma)$. We use partitioned DFAs to label elements of $L(\mathcal{B})$ with symbols from Γ. The language $L(\mathcal{A}_a)$ will be the set of a-labelled words. We do not introduce partitioned NFAs, since for NFAs the languages $L(\mathcal{A}_a)$ $(a \in \Gamma)$ would not partition $L(\mathcal{B})$ and thus, a word could get several labels (one could label words from $L(\mathcal{B})$ with subsets of Γ).

A (ε-free) *rational transducer* is a tuple $\mathcal{T} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q (the set of states), Σ (the input alphabet), and Γ (the output alphabet) are finite sets, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq Q \times \Sigma \times \Gamma^+ \times Q$ is a finite transition relation. A transition $(q, a, w, p) \in \delta$ is also written as $q \xrightarrow{a|w} p$. The rational transducer $\mathcal{T}$ defines a binary relation $[\![\mathcal{T}]\!] \subseteq \Sigma^* \times \Gamma^*$ in the usual way. For a language $L \subseteq \Sigma^*$ let

$$\mathcal{T}(L) = \{v \in \Gamma^* \mid \exists u \in L : (u, v) \in [\![\mathcal{T}]\!]\}.$$

2.3. 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 $a < c < b$. For $a \in A$, let $\mathsf{child}(a, T)$ (the set of children of a) be the set of all $b \in A$ such that $a \lessdot b$. The set of leaves of T is $\mathsf{leaf}(T) = \{a \in A \mid \mathsf{child}(a, T) = \emptyset\}$. For $a \in A$ let $T{\restriction}_a$ be the subtree of T rooted at a, i.e., the set of nodes of $T{\restriction}_a$ is $\{b \in A \mid a \leq b\}$. The tree T is *finitely branching* if $\mathsf{child}(a, T)$ is finite for all $a \in A$. An *infinite path* of T is an infinite chain $a_0 \lessdot a_1 \lessdot a_2 \lessdot \cdots$; *finite paths* are defined analogously. If T is finite and $a \in A$, then the *height* of a in T is the maximal length of a path that starts in a. For trees $T_1 = (A_1; \leq_1)$ and $T_2 = (A_2; \leq_2)$ we write $T_1 \cong T_2$ in case T_1 and T_2 are isomorphic, i.e., there exists a bijection $f : A_1 \to A_2$ such that for all $a, b \in A_1$: $a \leq_1 b$ if and only if $f(a) \leq_2 f(b)$.

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. Most of the time, we will identify the language L with the tree $(L; \leq_{\mathsf{pref}})$. Moreover, if $L = \mathsf{pref}(L)$ (i.e., L is prefix-closed), then T is a finitely branching tree.

A countable tree T is called *regular* if T has only finitely many subtrees up to isomorphism. 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 $\varepsilon \in L$. We require that the empty word ε belongs to L in order to ensure the existence of a root (otherwiese $(L; \leq_{\mathsf{pref}})$ would be only a forest). If L is accepted by the accessible DFA $\mathcal{A}$, 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.

Our 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 the 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 will be briefly considered in Section 4.9.

2.4. Linear orders

See [33] for a thorough introduction into linear orders. Let η be the order type of the rational numbers, ω 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. 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. Hence, for instance, η, ω, $\overline{\omega}$, and every finite linear order are regular linear orders.

For two linear orders $\Lambda_1 = (L_1; \leq_1)$ and $\Lambda_1 = (L_2; \leq_2)$ with $L_1 \cap L_2 = \emptyset$ we define the sum $\Lambda_1 + \Lambda_2 = (L_1 \cup L_2; \leq)$, 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 $y \in L_2$. We define the product $\Lambda_1 \cdot \Lambda_2 = (L_1 \times L_2; \leq)$ where $(x_1, x_2) \leq (y_1, y_2)$ if and only if either $x_2 <_2 y_2$ or ($x_2 = y_2$ and $x_1 \leq_1 y_1$).

An *interval* of Λ is a subset $I \subseteq L$ such that $x < z < y$ and $x, y \in I$ implies $z \in I$. An interval is *right-closed* (resp. *left-closed*) if it has a greatest (resp. smallest) element and it is *closed* if it is both right-closed and left-closed. An interval I is *dense* (resp., *scattered*) if the linear order $\leq$ restricted to I is dense (resp., scattered). A 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, a *predecessor* (resp., *successor*) of x need not exist, but if it exists then it is unique.

2.5. Generalized words

Generalized words are countable colored linear orders. Let Σ be a (possibly infinite) 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. As for trees, we write $u_1 \cong u_2$ for generalized words $u_1 = (L_1; \leq_1, \tau_1)$ and $u_2 = (L_2; \leq_2, \tau_2)$ in case u_1 and u_2 are isomorphic, i.e., there exists a bijection $f : L_1 \to L_2$ such that for all $a, b \in L_1$: $a \leq_1 b$ if and only if $f(a) \leq_2 f(b)$, and $\tau_1(a) = \tau_2(f(a))$.

Let $u = (L; \leq, \tau)$ be a generalized word over Σ with $\Gamma = \mathsf{alph}(u)$. Let $v_a = (L_a; \leq_a, \tau_a)$ be a generalized word for each $a \in \Gamma$. We define the generalized word $u[(a/v_a)_{a\in\Gamma}] = (L'; \leq, \tau')$ as follows:

- $L' = \{(x, y) \mid y \in L, x \in L_{\tau(y)}\}$,
- $(x, y) \leq (x', y')$ if and only if either $y < y'$ or ($y = y'$ and $x \leq_{\tau(y)} x'$), and
- $\tau'(x, y) = \tau_{\tau(y)}(x)$.

Thus, $u[(a/v_a)_{a\in\Gamma}]$ is obtained from u by replacing every a-labelled point by v_a (for all $a \in \Gamma$). Now we can define the regular operations on words. In order to do so we need the following words. The words ab and a^ω for $a, b \in \Sigma$ are as usual. The generalized word $a^{\overline{\omega}}$ has $\overline{\omega}$ as underlying order and every element is colored with a. Finally, we let $[a_1, \dots, a_n]^\eta$ be the generalized word with underlying order η where the coloring is such that any point is labeled by some a_i $(1 \leq i \leq n)$ and, moreover, for any two points $x < y$ and any $1 \leq i \leq n$ we find a point z with $x < z < y$ colored by a_i. It can be shown that this describes a unique word up to isomorphism [33].

Definition 3 (Regular Operations). *Let $u, v, u_1, \ldots, u_n$ be words over Σ. We let:*

$$\begin{aligned} uv &= (ab)[a/u, b/v] & u^{\omega} &= a^{\omega}[a/u] \\ [u_1, \ldots, u_n]^{\eta} &= [a_1, \ldots, a_n]^{\eta}[a_1/u_1, \ldots, a_n/u_n] & u^{\overline{\omega}} &= a^{\overline{\omega}}[a/u]. \end{aligned}$$

Thus, the underlying linear order of uv is the sum of the underlying linear orders of u and v. Intuitively, we have $u^{\omega} = uuu\cdots$ and $u^{\overline{\omega}} = \cdots uuu$. 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. The least set of words which is closed under the regular operations and contains the singleton words a for $a \in \Sigma$ is called the set of *regular words* over Σ, denoted $\mathsf{Reg}(\Sigma)$. Note that this implies that every regular word is non-empty, i.e., its domain is a non-empty set. Moreover, although we allow Σ to be infinite (this will be useful later), the alphabet $\mathsf{alph}(u)$ of a regular word u must be finite. Clearly, every regular word can be described by a *regular expression* over the above operations, but this regular expression is in general not unique.

Example 4. *Here are some typical identities between regular words, where X is a finite set of regular words, $n \geq 0$, $m \geq 1$, $u, u_1, \ldots, u_n \in X$, every v_i $(1 \leq i \leq m)$ has one of the forms X^{η}, yX^{η}, $X^{\eta}z$, $yX^{\eta}z$ with $y, z \in X$, and v, w are regular words:*

$$\begin{gathered} X^{\eta}X^{\eta} \cong X^{\eta}uX^{\eta} \cong (X^{\eta})^{\omega} \cong (X^{\eta}u)^{\omega} \cong (X^{\eta})^{\overline{\omega}} \cong (uX^{\eta})^{\overline{\omega}} \cong X^{\eta}, \\ [u_1, \ldots, u_n, v_1, \ldots, v_m]^{\eta} \cong X^{\eta}, \qquad (1) \\ (vw)^{\omega} = v(wv)^{\omega}, \quad (vw)^{\overline{\omega}} = (wv)^{\overline{\omega}}w. \end{gathered}$$

In (1) *it is crucial that $m > 0$. This allows to only require $\{u_1, \ldots, u_n\} \subseteq X$ instead of $\{u_1, \ldots, u_n\} = X$. A complete axiomatization of the equational theory of regular words can be found in [3].*

By a result of Heilbrunner [14], regular words can be characterized by partitioned DFAs as follows:[3] Let $\mathcal{A} = (Q, \Gamma, \delta, q_0, (F_a)_{a \in \Sigma})$ be a partitioned DFA,

[3] The notion of partitioned DFA is not used in [14] but the equivalence of partitioned DFAs and equational systems as used in [14] is obvious, see also [2].

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 (describing the regular word u) a partitioned DFA $\mathcal{A}$ with $u \cong w(\mathcal{A})$, see e.g. [36, proof of Proposition 2] for a simple construction. The other direction is more difficult. Heilbrunner has shown in [14] how to compute from a given partitioned DFA $\mathcal{A}$ (such that $w(\mathcal{A})$ is non-empty) a regular expression for the word $w(\mathcal{A})$, which is therefore regular.[4] Unfortunately, the size of the regular expression produced by Heilbrunner's algorithm is exponential in the size of $\mathcal{A}$. In Section 4.4, we will see that a succinct representation of a regular expression for $w(\mathcal{A})$ can be produced in polynomial time.

By replacing a symbol a (which w.l.o.g. is a natural number) by the order $\mathbf{a} + \overline{\omega} + \omega$, one can show that the isomorphism problem for regular words (given by partitioned DFAs) can be reduced (in logspace) to the isomorphism problem for regular linear orders (given by DFAs). In other words, node labels can be eliminated as for regular trees (as remarked at the end of Section 2.3). So, the reader might ask, why we consider the isomorphism problem for regular words and do not restrict to regular linear orders. The point is that even if we start with regular linear orders, in the course of our polynomial isomorphism check regular words will naturally arise.

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.

[4] In fact, Heilbrunner speaks about systems of equations and their least solutions instead of partitioned DFAs. But these two formalisms can be easily (and efficiently) transformed into each other.

3.1. Upper bounds

Theorem 5. *The following problem can be solved in polynomial time:*

INPUT: Two DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.
QUESTION: $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$?

Proof. By taking the disjoint union of $\mathcal{A}_1$ and $\mathcal{A}_2$, it suffices to solve the following problem in polynomial time:

INPUT: A DFA $\mathcal{A}$ without initial state and two final states p, q of $\mathcal{A}$.
QUESTION: $(L(\mathcal{A}, p); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}, q); \leq_{\mathsf{pref}})$?

Note that $\varepsilon \in L(\mathcal{A}, p) \cap L(\mathcal{A}, q)$ since p and q are final. Let $\mathcal{A} = (Q, \Sigma, \delta, F)$. In fact, we will compute in polynomial time the equivalence relation

$$\mathsf{iso} = \{(p, q) \in F \times F \mid (L(\mathcal{A}, p); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}, q); \leq_{\mathsf{pref}})\}.$$

This will be done similarly to the classical partition refinement algorithm for checking bisimilarity of finite state systems [18].

For $p \in F$ and $C \subseteq F$ let $L(\mathcal{A}, p, C)$ be the set of all words accepted by the DFA $(Q, \Sigma, \delta, p, C)$. Hence, the sets $L(\mathcal{A}, p, \{q\})$ $(q \in F)$ partition $L(\mathcal{A}, p)$. Let us say that a node $u \in L(\mathcal{A}, p)$ is of type q if $u \in L(\mathcal{A}, p, \{q\})$. For $p \in F$ and $C \subseteq F$ let us define the subset $K(\mathcal{A}, p, C) \subseteq L(\mathcal{A}, p, C)$ as the set of all non-empty words over Σ labeling a path from p to a state from C without intermediate final states; this is clearly a regular language and a DFA for $K(\mathcal{A}, p, C)$ can be easily computed in polynomial time from $\mathcal{A}$, p, and C: We take the DFA $\mathcal{A}$ and remove every transition leaving a final state from F. Moreover, we introduce a copy p' of p, which will be the new initial state and there is an a-labeled transition from p' to q if and only if there is an a-labeled transition from p to q in $\mathcal{A}$. Finally, C is the set of final states.

Note that if $u \in L(\mathcal{A}, p)$ is of type q, then the nodes uv with $v \in K(\mathcal{A}, q, F)$ are exactly the children of u in the tree $(L(\mathcal{A}, p); \leq_{\mathsf{pref}})$. Let $n(p, q) \in \mathbb{N} \cup \{\infty\}$ be the cardinality of the language $K(\mathcal{A}, p, \{q\})$. By Lemma 2, each of these numbers $n(p, q)$ can be computed in polynomial time. For $C \subseteq F$ let $n(p, C) = \sum_{q \in C} n(p, q)$. Thus $n(p, C)$ is the cardinality of the language $K(\mathcal{A}, p, C)$.

Let us now compute the equivalence relation iso. As already remarked, this will be done by a partition refinement algorithm. Assume that R is an equivalence relation on F. We define the new equivalence relation $\widetilde{R}$ on F as follows:

$$\widetilde{R} = \{(p, q) \in R \mid n(p, C) = n(q, C) \text{ for every equivalence class } C \text{ of } R\}.$$

Thus, $\widetilde{R}$ is a refinement of R which can be computed in polynomial time from R. Let us define a sequence of equivalence relations $R_0, R_1, \ldots$ on F as follows: $R_0 = F \times F$, $R_{i+1} = \widetilde{R_i}$. Then, there exists $k < |F|$ such that $R_k = R_{k+1}$. We claim that $R_k = \mathsf{iso}$. A simple argument shows that for every equivalence relation R on F with $\mathsf{iso} \subseteq R$, one has $\mathsf{iso} \subseteq \widetilde{R}$ as well. Hence, by induction over $i \geq 0$, one gets $\mathsf{iso} \subseteq R_i$ for all $i \geq 0$.

For the other direction, we show that if R is an equivalence relation on F with $R = \widetilde{R}$ (this holds for R_k), then $R \subseteq \mathsf{iso}$. So, assume that $(p_1, p_2) \in R = \widetilde{R}$. We define an isomorphism $f : (L(\mathcal{A}, p_1); \leq_{\mathsf{pref}}) \to (L(\mathcal{A}, p_2); \leq_{\mathsf{pref}})$ as the limit of isomorphisms f_n, $n \geq 1$. Here, f_n is an isomorphism between the trees that result from $(L(\mathcal{A}, p_1); \leq_{\mathsf{pref}})$ and $(L(\mathcal{A}, p_2); \leq_{\mathsf{pref}})$ by cutting off all nodes below level n (the roots are on level 1). Let us call these trees $(L(\mathcal{A}, p_i); \leq_{\mathsf{pref}})\restriction_n$ $(i \in \{1, 2\})$. Moreover, f_n has the additional property that if f_n maps a node u_1 of type q_1 to a node u_2 of type q_2, then we have $(q_1, q_2) \in R$. Assume that f_n is already constructed and let u_1 of type q_1 be a leaf of $(L(\mathcal{A}, p_1); \leq_{\mathsf{pref}})\restriction_n$. Let $u_2 = f(u_1)$ be of type q_2; it is a leaf of $(L(\mathcal{A}, p_2); \leq_{\mathsf{pref}})\restriction_n$. Then we have $(q_1, q_2) \in R = \widetilde{R}$ and hence for every equivalence class C of R we have $n(q_1, C) = n(q_2, C)$. We can therefore find a bijection g between the languages $K(\mathcal{A}, q_1, F)$ and $K(\mathcal{A}, q_2, F)$ such that for all $v \in K(\mathcal{A}, q_1, F)$ we have: If v (resp. $g(v)$) is of type r_1 (resp. r_2), then $(r_1, r_2) \in R$. Note that the nodes $u_i v$ with $v \in K(\mathcal{A}, q_i, F)$ are the children of u_i in the tree $(L(\mathcal{A}, p_1); \leq_{\mathsf{pref}})$. We now extend the isomorphism f_n by g and do this for all leaves u_1 of $(L(\mathcal{A}, p_1); \leq_{\mathsf{pref}})\restriction_n$. This gives us the isomorphism f_{n+1}. □

The above proof show that if $(L_1; \leq_{\mathsf{pref}}) \cong (L_2; \leq_{\mathsf{pref}})$ for regular languages L_1 and L_2, then there even exists a computable isomorphism. For this, we have to fix in the second part of the proof computable bijections between the languages $K(\mathcal{A}, q_1, C)$ and $K(\mathcal{A}, q_2, C)$ for every $(q_1, q_2) \in \mathsf{iso}$ and every equivalence class C of iso. We can, for instance, map the length-lexicographically i^{th} word from $K(\mathcal{A}, q_1, C)$ to the length-lexicographically i^{th} word from $K(\mathcal{A}, q_2, C)$. Note that in general there does not exist a polynomial time computable isomorphism since the output length may grow exponentially with the input length (e.g., if $L_1 = \{\varepsilon\} \cup \{a, b\}^* c$ and $L_2 = \{\varepsilon\} \cup a^* c$).

Corollary 6. *The following problem belongs to* $\mathsf{EXPTIME}$*:*

INPUT: Two NFAs $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *such that* $\varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$*.*
QUESTION: $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$*?*

Proof. In exponential time, we can transform $\mathcal{A}_1$ and $\mathcal{A}_2$ into DFAs using the powerset construction. Then we can apply Theorem 5. $\square$

Theorem 7. *The following problem belongs to* PSPACE*:*

INPUT: Two acyclic NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $\varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.
QUESTION: $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$?

Proof. By [25], isomorphism for finite trees, given explicitly by adjacency lists, can be decided in deterministic logspace. Hence, by Lemma 1 it suffices to show that for a given acyclic NFA, the adjacency list representation for the tree $(L(\mathcal{A}); \leq_{\mathsf{pref}})$ can be computed by a PSPACE-transducer. This is straightforward. Assume that Σ is the alphabet of $\mathcal{A}$ and that n is the number of states of $\mathcal{A}$. Let us fix an arbitrary order on Σ and let z be the largest symbol in Σ.

The language $L(\mathcal{A})$ only contains words of length at most $n-1$. In an outer loop we generate the language $L(\mathcal{A})$. For this, we enumerate all words (e.g. in lexicographic order) of length at most $n-1$ and test whether the current word is accepted by $\mathcal{A}$. For each enumerated word $u \in L(\mathcal{A})$, we have to output a list of all children of u in the tree $(L(\mathcal{A}); \leq_{\mathsf{pref}})$. In an inner loop, we enumerate (again in lexicographic order) all words uv ($v \in \Sigma^+$) of length at most $n-1$ and check whether $uv \in L(\mathcal{A})$. In case, we find such a word $uv \in L(\mathcal{A})$, we output uv and do the following: If $v \in \{z\}^+$, then the inner loop terminates. On the other hand, if $v = v'az^k$, where $a \neq z$, then we jump in the inner loop to the word $uv'b$, where b is the symbol following a in our order. $\square$

3.2. Lower bounds

The main result of this section states that the isomorphism problem for regular trees that are represented by NFAs is $\mathsf{EXPTIME}$-hard, which matches the upper bound from the previous section. It is straightforward to prove PSPACE-hardness by a reduction from universality for NFAs, which is PSPACE-complete [35]: Take an NFA $\mathcal{A}$ over a finite alphabet Σ and let $\# \notin \Sigma$. One can easily construct an NFA $\mathcal{A}'$ for the language $\Sigma^* \cup L(\mathcal{A})\#$. Moreover, we have $(L(\mathcal{A}'); \leq_{\mathsf{pref}}) \cong (\Sigma^* \cup \Sigma^*\#, \leq_{\mathsf{pref}})$ if and only if $L(\mathcal{A}) = \Sigma^*$.

The proof for the $\mathsf{EXPTIME}$ lower bound is more involved. Here is a rough outline: $\mathsf{EXPTIME}$ coincides with alternating polynomial space [6]. Checking whether a given input is accepted by a polynomial space bounded alternating Turing machine M amounts to evaluate a boolean circuit whose gates correspond to configurations of M. Using a construction from [16], the evaluation problem for (finite) boolean circuits can be reduced to the isomorphism problem for (finite) trees. In our case, the boolean circuit will be infinite. Nevertheless, the infinite boolean circuits we have to deal with can be evaluated because on every infinite path that starts in the root (the output gate) there will be 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 [16] to an infinite boolean circuit (that arises from our construction) will yield two infinite trees, which are isomorphic if and only if our boolean circuit evaluates to true. Luckily, these two trees turn out to be regular, and they can be represented by small NFAs.

3.2.1. Infinite boolean circuits.

Let us fix the alphabet

$$\Omega = \{a, \ell_\wedge, \ell'_\wedge, r_\wedge, \ell_\vee, \ell'_\vee, r_\vee\}. \tag{2}$$

In the following, we will only consider *prefix-closed* trees over the alphabet Ω (we will not mention this explicitly all the time). Moreover, we will identify the tree $(L; \leq_{\mathsf{pref}})$ with the language L. Now, consider such a tree $T \subseteq \Omega^*$. Then, T is *well-formed*, if the following conditions hold:

(a) If $u = \varepsilon$ or $u \in T$ ends with $\ell_\vee$, $\ell_\wedge$, $r_\vee$, or $r_\wedge$, then $\mathsf{child}(u, T)$ is one of the following sets, where $\circ \in \{\vee, \wedge\}$: $\{u\,\ell_\circ, u\,r_\circ\}$, $\{u\,\ell'_\circ, u\,r_\circ\}$, $\{ua, u\,\ell'_\circ, u\,r_\circ\}$.

(b) If $u \in T$ ends with a, $\ell'_\vee$, or $\ell'_\wedge$, then u is a leaf of T.

(c) For every infinite path P in T that starts in the root, there exists $u \in P$ with $ua \in T$.

Note that a well-formed tree T is always infinite; it contains an infinite path of the form $r_1 r_2 r_3 \cdots$, where $r_i \in \{r_\wedge, r_\vee\}$ for all $i \geq 1$. Let us define the set

$$\mathsf{cut}(T) = \{u \in T \mid ua \in T,\ \forall v <_{\mathsf{pref}} u : va \notin T\}. \tag{3}$$

Hence, on every infinite path in T there is a unique node from $\mathsf{cut}(T)$.

With a well-formed tree T we associate an infinite tree-like boolean circuit $\mathsf{bool}(T)$ as follows: The gates of $\mathsf{bool}(T)$ are the nodes of T that do not end with a.

- The set of input gates for $u \in T$ is $\mathsf{child}(u, T) \setminus \{ua\}$.

- If $ur_\vee \in T$ (resp. $ur_\wedge \in T$), then u is an or-gate (resp. and-gate).
- If $u\ell'_\wedge \in T$ and $ua \notin T$, then $u\ell'_\wedge$ is a true-gate.
- If $u\ell'_\wedge \in T$ and $ua \in T$, then $u\ell'_\wedge$ is a false-gate.
- If $u\ell'_\vee \in T$ and $ua \notin T$, then $u\ell'_\vee$ is a false-gate.
- If $u\ell'_\vee \in T$ and $ua \in T$, then $u\ell'_\vee$ is a true-gate.

Although $\mathsf{bool}(T)$ is an infinite boolean circuit, the fact that T is well-formed ensures that the root of $\mathsf{bool}(T)$ can be evaluated: We simply remove from T all nodes that have a proper prefix from $\mathsf{cut}(T)$. The resulting tree has no infinite path and since it is finitely branching it is finite by König's lemma. If $u \in \mathsf{cut}(T)$ is such that $u\ell'_\wedge \in T$ (resp., $u\ell'_\vee \in T$), then u can be transformed into a false-gate (resp., true-gate). Then, one has to evaluate the resulting finite boolean circuit.

We next transform a tree $T \subseteq \Omega^*$ into trees $[T]_1, [T]_2 \subseteq \{\ell, r\}^*$ using two rational transducers. These two transducers only differ in their initial state. For $i \in \{1, 2\}$, let $\mathcal{T}_i$ be the transducer from Figure 1, where the initial state is q_i and all states are final. Then, for a tree $T \subseteq \Omega^*$ and $i \in \{1, 2\}$ let $[T]_i = \mathsf{pref}(\mathcal{T}_i(T))$. We will show that for every well-formed tree $T \subseteq \Omega^*$: $\mathsf{bool}(T)$ evaluates to true if and only if $[T]_1 \cong [T]_2$. (Lemma 13) For this, we first have to show a few lemmas. In Figures 2–7 below, the edge labels ℓ and r are added in order to make the construction clearer. These edge labels do not have to be preserved by isomorphisms. Hence, for checking whether two trees are isomorphic, the edge labels ℓ and r should be ignored.

Lemma 8. *Let $T = \{\varepsilon, \ell'_\vee\} \cup r_\vee U$ or $T = \{\varepsilon, \ell'_\wedge\} \cup r_\wedge U$ for a tree U (hence, also T is a tree). Then $[T]_1 \cong [T]_2$ if and only if $[U]_1 \cong [U]_2$.*

Proof. We first consider the case $T = \{\varepsilon, \ell'_\vee\} \cup r_\vee U$. Let us compute compute $\mathcal{T}_1(T)$ and $\mathcal{T}_2(T)$. We have

$$\mathcal{T}_1(\ell'_\vee) = \mathcal{T}_2(\ell'_\vee) = \{\ell^2, r\ell^2\}. \tag{4}$$

Next, we have to compute $\mathcal{T}_1(r_\vee U)$. There are two transitions starting in q_1, where $r_\vee$ can be read, namely

$$q_1 \xrightarrow{r_\vee | \ell r \ell} q_2 \quad \text{and} \quad q_1 \xrightarrow{r_\vee | r^2 \ell} q_1.$$

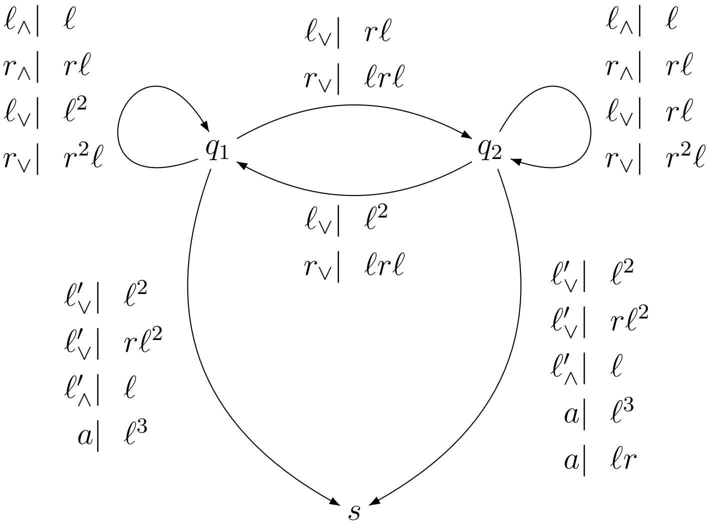


Figure 1: The transducer

Hence, we get

$$\mathcal{T}_1(r_\vee U) = r^2\ell\,\mathcal{T}_1(U) \cup \ell r\ell\,\mathcal{T}_2(U). \tag{5}$$

Similarly, we get

$$\mathcal{T}_2(r_\vee U) = \ r^2\ell\,\mathcal{T}_2(U) \cup \ell r\ell\,\mathcal{T}_1(U). \tag{6}$$

From (4), (5), and (6) it follows that the trees $[T]_i = \mathsf{pref}(\mathcal{T}_i(\{\varepsilon, \ell'_\vee\} \cup r_\vee U))$ ($i \in \{1,2\}$) are the ones shown in Figure 2. The equivalence of $[T]_1 \cong [T]_2$ and $[U]_1 \cong [U]_2$ is obvious from these diagrams.

Let us now consider the case $T = \{\varepsilon, \ell'_\wedge\} \cup r_\wedge U$. We have $\mathcal{T}_1(\ell'_\wedge) = \mathcal{T}_2(\ell'_\wedge) = \{\ell\}$, $\mathcal{T}_1(r_\wedge U) = r\ell\mathcal{T}_1(U)$, and $\mathcal{T}_2(r_\wedge U) = r\ell\mathcal{T}_2(U)$. The trees $[T]_i = \mathsf{pref}(\mathcal{T}_i(\{\varepsilon, \ell'_\wedge\} \cup r_\wedge U))$ ($i \in \{1,2\}$) are the ones shown in Figure 3. The equivalence of $[T]_1 \cong [T]_2$ and $[U]_1 \cong [U]_2$ is again obvious from these diagrams. □

The following three lemmas can be shown with the same kinds of arguments as for Lemma 8. We therefore only sketch the proofs.

Lemma 9. *Let $T = \{\varepsilon, \ell'_\vee, a\} \cup r_\vee U$ for a tree U (hence, also T is a tree). Then $[T]_1 \cong [T]_2$.*

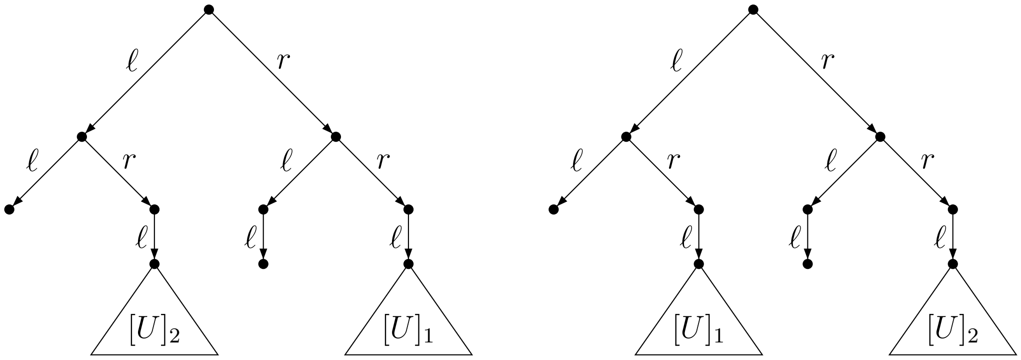


Figure 2: $[T]_1$ (left) and $[T]_2$ (right) from Lemma 8 for $\vee$

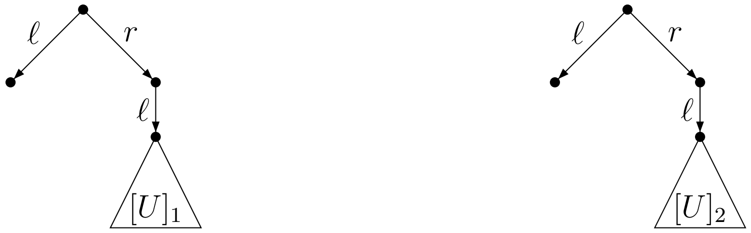


Figure 3: $[T]_1$ (left) and $[T]_2$ (right) from Lemma 8 for $\wedge$

Proof. We have $\mathcal{T}_1(a) = \{\ell^3\}$ and $\mathcal{T}_2(a) = \{\ell^3, \ell r\}$. It follows, that the trees $[T]_1$ and $[T]_2$ are as shown in Figure 4. Clearly, we have $[T]_1 \cong [T]_2$. □

Lemma 10. *Let $T = \{\varepsilon, \ell'_\wedge, a\} \cup r_\wedge U$ for a tree U (hence, also T is a tree). Then $[T]_1 \not\cong [T]_2$.*

Proof. The trees $[T]_1$ and $[T]_2$ are shown in Figure 5. Clearly, we have $[T]_1 \not\cong [T]_2$. □

Lemma 11. *Let $T = \{\varepsilon\} \cup \ell_\vee U \cup r_\vee V$ for well-formed trees U, V (hence, also T is well-formed). Then $[T]_1 \cong [T]_2$ if and only if ($[U]_1 \cong [U]_2$ or $[V]_1 \cong [V]_2$).*

Proof. The trees $[T]_1$ and $[T]_2$ are shown in Figure 6. Since U and V are well-formed, in each of the trees $[U]_1$, $[U]_2$, $[V]_1$, and $[V]_2$, the root has two children. It follows easily that $[T]_1 \cong [T]_2$ if and only if ($[U]_1 \cong [U]_2$ or $[V]_1 \cong [V]_2$). □

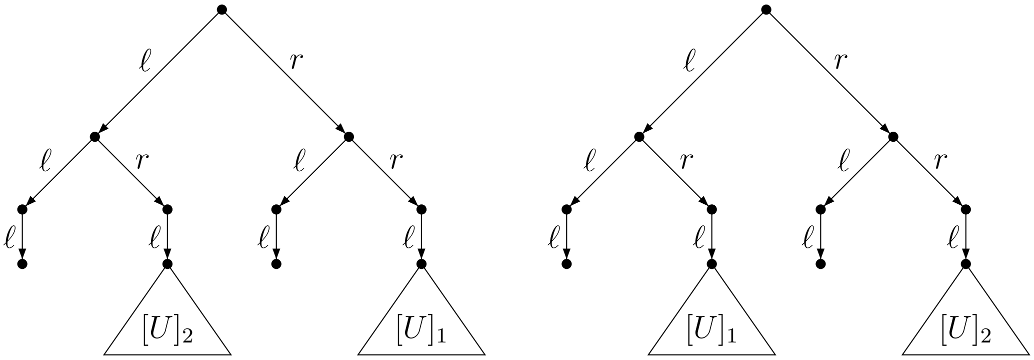


Figure 4: $[T]_1$ (left) and $[T]_2$ (right) from Lemma 9

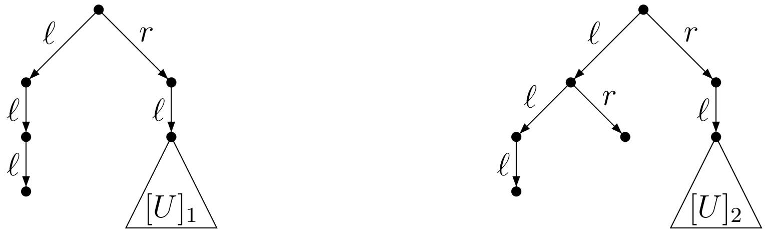


Figure 5: $[T]_1$ (left) and $[T]_2$ (right) from Lemma 10

Lemma 12. *Let $T = \{\varepsilon\} \cup \ell_\wedge U \cup r_\wedge V$ for well-formed trees U, V (hence, also T is well-formed). Then $[T]_1 \cong [T]_2$ if and only if ($[U]_1 \cong [U]_2$ and $[V]_1 \cong [V]_2$).*

Proof. The trees $[T]_1$ and $[T]_2$ are as shown in Figure 7. Since U and V are well-formed, in each of the trees $[U]_1$, $[U]_2$, $[V]_1$, and $[V]_2$, the root has two children. It follows easily that $[T]_1 \cong [T]_2$ if and only if ($[U]_1 \cong [U]_2$ and $[V]_1 \cong [V]_2$). □

Lemma 13. *For every well-formed tree $T \subseteq \Omega^*$ we have: $\mathsf{bool}(T)$ evaluates to true if and only if $[T]_1 \cong [T]_2$.*

Proof. Recall the definition of the set $\mathsf{cut}(T)$ from (3). From the definition it follows that $\mathsf{pref}(\mathsf{cut}(T))$ is a finitely branching tree without infinite paths. Hence, by König's lemma it is finite. Moreover, for every

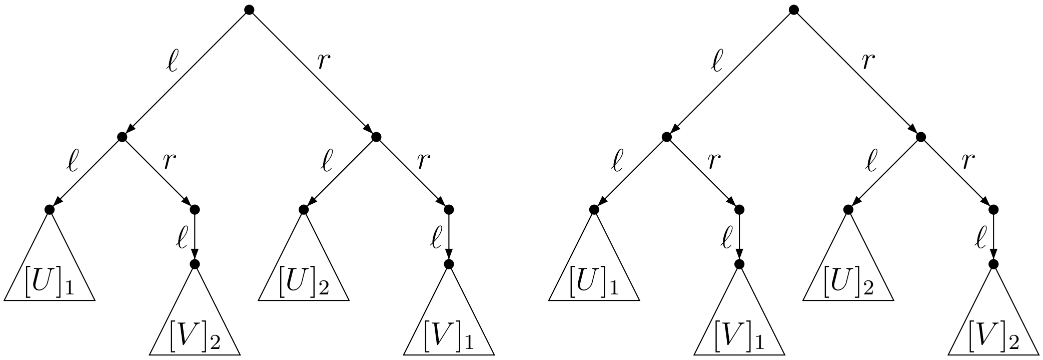


Figure 6: $[T]_1$ (left) and $[T]_2$ (right) from Lemma 11

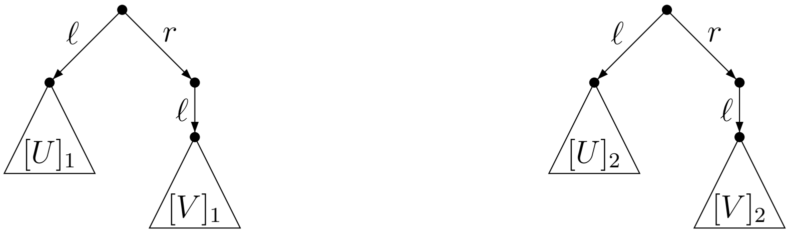


Figure 7: $[T]_1$ (left) and $[T]_2$ (right) from Lemma 12

$u \in \mathsf{pref}(\mathsf{cut}(T))$, the subtree $T{\upharpoonright_u}$ is well-formed as well (since $\mathsf{pref}(\mathsf{cut}(T)) \subseteq \{\varepsilon\} \cup \Omega^*\{\ell_\vee, \ell_\wedge, r_\vee, r_\wedge\}$). Inductively over the height of $u \in \mathsf{pref}(\mathsf{cut}(T))$ in the finite tree $\mathsf{pref}(\mathsf{cut}(T))$, we will prove for every $u \in \mathsf{pref}(\mathsf{cut}(T))$: $[T{\upharpoonright_u}]_1 \cong [T{\upharpoonright_u}]_2$ if and only if $\mathsf{bool}(T{\upharpoonright_u})$ evaluates to true.

For the induction base, let $u \in \mathsf{cut}(T)$ be a leaf of $\mathsf{pref}(\mathsf{cut}(T))$. Hence, we have $ua \in T$. If $u\ell'_\wedge \in T$, then in $\mathsf{bool}(T{\upharpoonright_u})$, the root is an and-gate for which one of the inputs (namely $u\ell'_\wedge$) is a false-gate. Hence, $\mathsf{bool}(T{\upharpoonright_u})$ evaluates to false. Moreover, Lemma 10 implies that $[T{\upharpoonright_u}]_1 \not\cong [T{\upharpoonright_u}]_2$. On the other hand, if $u\ell'_\vee \in T$, then in $\mathsf{bool}(T{\upharpoonright_u})$, the root is an or-gate for which one of the inputs (namely $u\ell'_\vee$) is a true-gate. Hence, $\mathsf{bool}(T{\upharpoonright_u})$ evaluates to true. Moreover, Lemma 9 implies that $[T{\upharpoonright_u}]_1 \cong [T{\upharpoonright_u}]_2$. This concludes the induction base.

Next, let $u \in \mathsf{pref}(\mathsf{cut}(T))$ be a proper prefix of a node from $\mathsf{cut}(T)$. In particular $u \notin \mathsf{cut}(T)$. We can distinguish 4 different cases:

Case 1. $\mathsf{child}(u,T) = \{u\ell_\wedge, ur_\wedge\}$. We must have $\{u\ell_\wedge, ur_\wedge\} \subseteq \mathsf{pref}(\mathsf{cut}(T))$. Hence, the induction hypothesis (IH) holds for $u\ell_\wedge$ and $ur_\wedge$. We get:

$$\begin{aligned}
\mathsf{bool}(T\restriction_u) \text{ evaluates to } \mathsf{true} \quad &\iff \quad \mathsf{bool}(T\restriction_{u\ell_\wedge}) \text{ evaluates to } \mathsf{true} \text{ and} \\
& \quad \mathsf{bool}(T\restriction_{ur_\wedge}) \text{ evaluates to } \mathsf{true} \\
&\overset{\text{(IH)}}{\iff} \quad [T\restriction_{u\ell_\wedge}]_1 \cong [T\restriction_{u\ell_\wedge}]_2 \text{ and} \\
& \quad [T\restriction_{ur_\wedge}]_1 \cong [T\restriction_{ur_\wedge}]_2 \\
&\overset{\text{Lemma 12}}{\iff} \quad [T\restriction_u]_1 \cong [T\restriction_u]_2
\end{aligned}$$

Case 2. $\mathsf{child}(u,T) = \{u\ell_\vee, ur_\vee\}$. This case is analogous to Case 1, using Lemma 11.

Case 3. $\mathsf{child}(u,T) = \{u\ell'_\wedge, ur_\wedge\}$. Since $u \notin \mathsf{cut}(T)$, we have $ua \notin T$. We must have $ur_\wedge \in \mathsf{pref}(\mathsf{cut}(T))$. Moreover, in $\mathsf{bool}(T\restriction_u)$, the root is an and-gate, where one of the inputs is a true-gate and the other input is the root for the boolean circuit $\mathsf{bool}(T\restriction_{ur_\wedge})$. Hence, we get:

$$\begin{aligned}
\mathsf{bool}(T\restriction_u) \text{ evaluates to } \mathsf{true} \quad &\iff \quad \mathsf{bool}(T\restriction_{ur_\wedge}) \text{ evaluates to } \mathsf{true} \\
&\overset{\text{(IH)}}{\iff} \quad [T\restriction_{ur_\wedge}]_1 \cong [T\restriction_{ur_\wedge}]_2 \\
&\overset{\text{Lemma 8}}{\iff} \quad [T\restriction_u]_1 \cong [T\restriction_u]_2
\end{aligned}$$

Case 4. $\mathsf{child}(u,T) = \{u\ell'_\vee, ur_\vee\}$. This case is analogous to Case 3. □

Our last auxiliary lemma states that an NFA for the tree $[L]_i$ can be easily computed from an NFA for L.

Lemma 14. *There is a logspace machine that computes from a given prefix-closed NFA $\mathcal{A}$ over the alphabet Ω a prefix-closed NFA $\mathcal{B}$ such that $L(\mathcal{B}) = [L(\mathcal{A})]_i$ for $i \in \{1,2\}$.*

Proof. Let $\mathcal{A} = (Q, \Omega, \delta, p_0, Q)$. Recall that all states of $\mathcal{T}_i$ and $\mathcal{A}$ are final. The prefix-closed NFA $\mathcal{B}$ is obtained from the direct product of $\mathcal{A}$ and $\mathcal{T}_i$ by adding further states so that every transition is labeled with a single symbol. Thus, the set of states of $\mathcal{B}$ contains $Q \times \{q_1, q_2, s\}$ and the initial state of $\mathcal{B}$ is (p_0, q_i). If $q \xrightarrow{b} q'$ in $\mathcal{A}$ and $t \xrightarrow{b|w} t'$ in $\mathcal{T}_i$ for $w \in \{\ell, r\}^+$, then we add $|w| - 1$ many new states to $\mathcal{B}$, which built up a w-labeled path from from (q,t) to (q',t'). □

3.2.2. EXPTIME*-hardness.*

We are now in the position to prove the main result of this section.

Theorem 15. *The following problem is* EXPTIME*-hard (and hence* EXPTIME*-complete):*

INPUT: Two prefix-closed NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$.
QUESTION: $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$?

Proof. The upper bound is stated in Corollary 6. For the lower bound we use the fact that EXPTIME equals the class of all sets that can be accepted in polynomial space on an alternating Turing machine [6]. Hence, let M be a polynomial space bounded alternating Turing machine such that the accepted language $L(M) \subseteq \{0,1\}^*$ is EXPTIME-complete. By adding a counter, one can assume that M has no infinite computation paths. By padding inputs, one can moreover assume that M works in space n for an input of length n. Let $Q = Q_\exists \cup Q_\forall$ be the set of states of M and let $\Gamma \supseteq \{0,1\}$ be the tape alphabet. W.l.o.g. we can assume that in every computation step, M moves from an existential state to a universal state or vice versa, and that the initial state q_0 is universal.

Let us now fix an input $w \in \{0,1\}^*$ of length n. We construct two prefix-closed NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $w \in L(M)$ if and only if

$$(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}}).$$

Let $\Theta = \Gamma \cup Q$. As usual, a configuration of M can be represented by a string from the language Θ^{n+1} (more precisely, from $\bigcup_{j=0}^{n-1} \Gamma^j Q \Gamma^{n-j}$). A word $u \in \Theta^*$ is a *valid computation of M on input w* if u is of the form $c_1 \cdots c_m$ for some $m \geq 0$ such that the following hold:

- $c_i \in \bigcup_{j=0}^{n-1} \Gamma^j Q \Gamma^{n-j}$ for all $1 \leq i \leq m$
- $c_i \vdash_M c_{i+1}$ (i.e., c_{i+1} is a successor configuration of c_i) for all $1 \leq i \leq m-1$
- $q_0 w \vdash_M c_1$

Note that ε is a valid computation in this sense. It is well known that from w one can construct in logspace a coaccessible NFA $\mathcal{A}_w$ such that $\mathcal{A}_w$ accepts all words over Θ that are *not* a valid computation of M on w [35].

Next, we will define a regular well-formed tree $T_w \subseteq \Omega^*$ (depending only on w) such that $\mathsf{bool}(T_w)$ evaluates to true if and only if $w \in L(M)$. Roughly, the idea is that T_w consists of all prefixes of sequences of configurations of length n, where symbols from Θ are coded by binary strings. Moreover, if such a sequence s is not a valid computation then we also include the word sa, where a is a new symbol, into the tree.

In the following, we identify the symbols in Θ with the integers $0, \dots, |\Theta|-1$ in an arbitrary way. We can assume that $|\Theta| \geq 2$. We define two morphisms

$$\varphi_\wedge : \Theta^* \to \{\ell_\wedge, r_\wedge\}^*$$
$$\varphi_\vee : \Theta^* \to \{\ell_\vee, r_\vee\}^*$$

as follows ($\circ \in \{\wedge, \vee\}$):

$$\varphi_\circ(a) \quad = \quad \begin{cases} r_\circ^a \ell_\circ & \text{if } 0 \leq a < |\Theta| - 1 \\ r_\circ^a & \text{if } a = |\Theta| - 1 \end{cases}$$

For $i \geq 1$, let φ_i be the mapping $\varphi_\wedge$ (resp. $\varphi_\vee$) if i is odd (resp., even). Similarly, for $x \in \{\ell, \ell', r\}$, let x_i be $x_\wedge$ (resp. $x_\vee$) if i is odd (resp., even). Then, the tree $T_w \subseteq \Omega^*$ is $\mathsf{pref}(T'_w)$, where

$$\begin{aligned} T'_w = &\left\{ \left(\prod_{i=1}^{m} r_i \varphi_i(c_i) \right) \ell'_{m+1} \mid m \geq 0, c_1, \dots, c_m \in \Theta^{n+1} \right\} \cup \\ &\left\{ \left(\prod_{i=1}^{m} r_i \varphi_i(c_i) \right) a \mid m \geq 0, c_1, \dots, c_m \in \Theta^{n+1}, c_1 \cdots c_m \in L(\mathcal{A}_w) \right\}. \end{aligned}$$

Clearly, T_w is regular, and a prefix-closed NFA for T_w can be computed in logspace from w (using the logspace computable coaccessible NFA $\mathcal{A}_w$).

Claim 1: T_w is well-formed.

Proof of Claim 1: The first two conditions (a) and (b) for well-formed trees are easy to check. For condition (c), we have to consider an arbitrary infinite path P of T_w and show that there exists $u \in P$ such that $ua \in T_w$. The infinite path P corresponds to an infinite word $r_1\varphi_1(c_1)r_2\varphi_2(c_2)r_3\varphi_2(c_3)\cdots$ with $c_i \in \Theta^{n+1}$ for all $i \geq 1$. Since M does not have infinite computations, there exists $m \geq 1$ such that $c_1 \cdots c_m$ is not a valid computation of M on input w. Hence, $c_1 \cdots c_m \in L(\mathcal{A}_w)$, which implies that

$$\left(\prod_{i=1}^{m} r_i \varphi_i(c_i) \right) a \in T_w.$$

Since $\prod_{i=1}^{m} r_i\varphi_i(c_i)$ belongs to the path P, this shows Claim 1.

Claim 2: $w \in L(M)$ if and only if $\mathsf{bool}(T_w)$ evaluates to true.

Proof of Claim 2: Let us consider the *finite* tree $\mathsf{pref}(\mathsf{cut}(T_w))$. For every node

$$g = r_\wedge\varphi_\wedge(c_1)r_\vee\varphi_\vee(c_2)r_\wedge \cdots \varphi_{m-1}(c_{m-1})r_m\varphi_m(c_m) \in \mathsf{pref}(\mathsf{cut}(T_w))$$

with $m \geq 0$ and $c_1, \ldots, c_m \in \Theta^{n+1}$ we will prove (by induction on the height of g) the following: If $c_1 \cdots c_m$ is a valid computation of M on input w, then c_m is an accepting configuration if and only if g evaluates to true in $\mathsf{bool}(T_w)$. Here, for $m = 0$, we define c_0 as the initial configuration q_0w.

So, assume that $g \in \mathsf{pref}(\mathsf{cut}(T_w))$ is of the above form and that $c_1 \cdots c_m$ is a valid computation of M on input w. We only consider that case that m is odd; the case that m is even can be dealt analogously. Thus,

$$g = r_\wedge\varphi_\wedge(c_1)r_\vee\varphi_\vee(c_2)r_\wedge \cdots \varphi_\vee(c_{m-1})r_\wedge\varphi_\wedge(c_m).$$

Then, in $\mathsf{bool}(T_w)$, the input gates for the or-gate g are $g\ell'_\vee$ and $gr_\vee$. Since $c_1 \cdots c_m$ is a valid computation of M on input w, ga does not belong to the tree T_w. Hence, in $\mathsf{bool}(T_w)$, $g\ell'_\vee$ is a false-gate. Thus, g evaluates to true if and only if $gr_\vee$ evaluates to true. From the structure of T_w we see that the latter holds if and only if there exists $c_{m+1} \in \Theta^{n+1}$ such that $gr_\vee\varphi_\vee(c_{m+1})$ evaluates to true. First assume that c_{m+1} is such that $c_1 \cdots c_m c_{m+1}$ is not a valid computation. The inputs for the and-gate $gr_\vee\varphi_\vee(c_{m+1})$ are $gr_\vee\varphi_\vee(c_{m+1})\ell'_\wedge$ and $gr_\vee\varphi_\vee(c_{m+1})r_\wedge$. Since $c_1 \cdots c_m c_{m+1}$ is not a valid computation, the word $gr_\vee\varphi_\vee(c_{m+1})a$ belongs to the tree T_w. Thus, in $\mathsf{bool}(T_w)$, $gr_\vee\varphi_\vee(c_{m+1})\ell'_\wedge$ is a false-gate and $gr_\vee\varphi_\vee(c_{m+1})$ evaluates to false. This holds for all c_{m+1} such that $c_1 \cdots c_m c_{m+1}$ is not a valid computation. Hence, $gr_\vee$ evaluates to true if and only if there exists a configuration $c_{m+1} \in \Theta^{n+1}$ such that $c_1 \cdots c_m c_{m+1}$ is a valid computation (which means that c_{m+1} is a successor configuration of c_m) and $gr_\vee\varphi_\vee(c_{m+1})$ evaluates to true in $\mathsf{bool}(T_w)$. Now, if $c_1 \cdots c_m c_{m+1}$ is a valid computation, then by induction, $gr_\vee\varphi_\vee(c_{m+1})$ (which belongs to $\mathsf{pref}(\mathsf{cut}(T_w))$ as well) evaluates to true in $\mathsf{bool}(T_w)$ if and only if c_{m+1} is an accepting configuration of M.

We have shown that g evaluates to true if and only if c_m has an accepting successor configuration. Finally, since m is odd, c_m is an existential configuration (recall that the initial configuration $c_0 = q_0w$ is universal). Thus, indeed, g evaluates to true if and only if c_m is accepting. Claim 2 follows by taking $m = 0$: $\mathsf{bool}(T_w) = \mathsf{true}$ if and only if ε evaluates to true in $\mathsf{bool}(T_w)$ if and only if c_0 is an accepting configuration of M if and only if $w \in L(M)$.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the rational transducers from Section 3.2.1. Using Lemma 14 we can compute in logspace from a prefix-closed NFA for T_w two prefix-closed NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $L(\mathcal{A}_i) = [T_w]_i$ for $i \in \{1, 2\}$. We have

$$\begin{aligned} w \in L(M) \quad &\overset{\text{Claim 2}}{\Longleftrightarrow} \quad \mathsf{bool}(T_w) \text{ evaluates to } \mathsf{true} \\ &\overset{\text{Lemma 13}}{\Longleftrightarrow} \quad (L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}}). \end{aligned}$$

This concludes the proof of the $\mathsf{EXPTIME}$ lower bound. □

3.2.3. PSPACE*-hardness.*

A variation of the proof for the $\mathsf{EXPTIME}$ lower bound shows:

Theorem 16. *The following problem is* PSPACE*-hard (and therefore* PSPACE*-complete):*

INPUT: Two prefix-closed acyclic NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$.
QUESTION: $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$?

Proof. The upper bound is stated in Theorem 7. For the lower bound, we use the same idea as in the proof of Theorem 15. In fact, we will use most of the notations from that proof; some of them will be slightly modified. This time, we use the fact that PSPACE equals the class of all sets that can be accepted in polynomial time on an alternating Turing machine. Hence, let M be a polynomial time bounded alternating Turing machine such that the accepted language $L(M) \subseteq \{0, 1\}^*$ is PSPACE-complete. Let $p(n)$ (a polynomial) be the time bound and let $q(n) = p(n) + 1$. We can assume that $q(n)$ is odd for all $n \geq 0$. W.l.o.g. we can assume again that M works in space n for an input of length n. Let $w \in \{0, 1\}^*$ be an input for M of length n.

Let us add to the alphabet Ω in (2) an additional symbol $r'_\vee$. The notions from Section 3.2.1 have to be extended to this new alphabet Ω. In condition (a) for the definition of a well-formed tree T, we also allow the set $\{ua, u\ell'_\vee, ur'_\vee\}$ for $\mathsf{child}(u, T)$. Moreover, every node $ur'_\vee \in T$ is a leaf of T. For the new definition of the set $\mathsf{cut}(T)$ we can reuse (3). Also the boolean circuit $\mathsf{bool}(T)$ can be defined as in Section 3.2.1; the truth value of a leaf ending with $r'_\vee$ is set arbitrarily (say true). Finally, let us extend the two transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ such that from q_1 and q_2 they can read the new symbol $r'_\vee$ and output ℓr and then terminate in the sink state s. Note that this ensures that

$$[\{\varepsilon, a, \ell'_\vee, r'_\vee\}]_1 = \{\varepsilon, \ell, \ell^2, \ell^3, \ell r, r, r\ell, r\ell^2\} = [\{\varepsilon, a, \ell'_\vee, r'_\vee\}]_2, \tag{7}$$

where we define again $[L]_i = \mathsf{pref}(\mathcal{T}_i(L))$.

We now define the well-formed tree $U_w \subseteq \Omega^*$ as $U_w = \mathsf{pref}(U'_w)$, where:

$$\begin{aligned} U'_w = &\left\{ \left(\prod_{i=1}^{m} r_i \varphi_i(c_i) \right) \ell'_{m+1} \mid 0 \leq m \leq q(n), c_1, \ldots, c_m \in \Theta^{n+1} \right\} \cup \\ &\left\{ \left(\prod_{i=1}^{m} r_i \varphi_i(c_i) \right) a \mid 0 \leq m \leq q(n), c_1, \ldots, c_m \in \Theta^{n+1}, c_1 \cdots c_m \in L(\mathcal{A}_w) \right\} \cup \\ &\left\{ \left(\prod_{i=1}^{q(n)} r_i \varphi_i(c_i) \right) r'_\vee \mid c_1, \ldots, c_{q(n)} \in \Theta^{n+1} \right\}. \end{aligned}$$

Note that U_w is finite. An acyclic prefix-closed NFA for U_w can be produced in logspace from w. Moreover, since every word from $\Theta^{(n+1)q(n)}$ is not a valid computation (since M terminates after $\leq p(n) = q(n) - 1$ steps), the boolean circuits $\mathsf{bool}(U_w)$ and $\mathsf{bool}(T_w)$ (where T_w was defined in the proof of Theorem 15) evaluate to the same truth value. Hence, using Claim 2 from the proof of Theorem 15, it follows that $w \in L(M)$ if and only if $\mathsf{bool}(U_w)$ evaluates to true. Using an analogon of Lemma 13 (whose proof uses fact (7)), this holds if and only if $[U_w]_1 \cong [U_w]_2$. Acyclic NFAs for $[U_w]_1$ and $[U_w]_2$ can be easily constructed in logspace from w (using an acyclic NFA for U_w). This concludes the proof of the theorem. □

3.2.4. P*-hardness.*

Theorem 17. *The following problem is* P*-hard (and hence* P*-complete):*

INPUT: Two prefix-closed acyclic DFAs $\mathcal{A}_1$ *and* $\mathcal{A}_2$.
QUESTION: $(L(\mathcal{A}_1); \leq_{\mathsf{pref}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{pref}})$*?*

Proof. The upper bound is stated in Theorem 5. For the lower bound, we reduce the P-complete monotone circuit value problem [13] to the problem from the theorem. Note that the tree $(L(\mathcal{A}); \leq_{\mathsf{pref}})$, where $\mathcal{A}$ is a prefix-closed

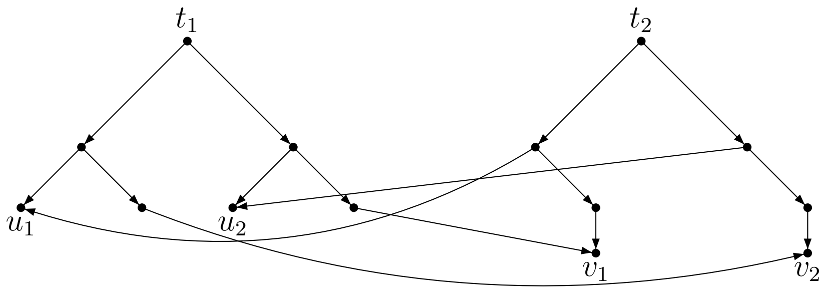


Figure 8: The or-construction in the proof of Theorem 17

acyclic DFA, is just the unfolding[5] of the underlying dag (directed acyclic graph) in the initial state of $\mathcal{A}$. Vice versa, from a dag D with a root node r one can construct a prefix-closed acyclic DFA $\mathcal{A}$ such that $(L(\mathcal{A}); \leq_{\mathsf{pref}})$ is isomorphic to the unfolding of D in r (let us denote the latter tree by $\mathsf{unfold}(D, r)$). One only has to associate labels to the edges of D. Hence, it suffices to construct from a given monotone circuit C a dag D which contains for every gate g of C two nodes g_1, g_2 such that g evaluates to true if and only if $\mathsf{unfold}(D, g_1) \cong \mathsf{unfold}(D, g_2)$. This is straightforward for the input gates of C. For and- and or-gates of C, we can use again the construction of [16]. Take the constructions from Figure 6 and 7, where in Figure 6 each of the subtrees $[U]_1$, $[U]_2$, $[V]_1$, and $[V]_2$ is represented only once. The construction for or-gates is shown in Figure 8. Unfolding the dag from Figure 8 in the nodes t_1 and t_2 and thereby duplicating u_1, u_2, v_1, and v_2 gives the trees from Figure 6. Assume that the dag D below the nodes u_1, u_2, v_1, and v_2 is already constructed. Here u_1 and u_2 correspond to a gate u and v_1 and v_2 correspond to a gate v. Hence, u (resp., v) evaluates to true if and only if $\mathsf{unfold}(D, u_1) \cong \mathsf{unfold}(D, u_2)$ (resp., $\mathsf{unfold}(D, v_1) \cong \mathsf{unfold}(D, v_2)$). Let t be an or-gate with inputs u and v. We add the nodes and edges as shown in Figure 8. Then the arguments from the proof of Lemma 11 show that u or v evaluates to true if and only if $\mathsf{unfold}(D, t_1) \cong \mathsf{unfold}(D, t_2)$. □

[5] The unfolding of a directed graph $G = (V, E)$ in a node $v \in V$ is the (in general infinite) tree $(P; \leq)$, where P is the set of all finite paths of G that start in node v. For finite paths p and p' we set $p \leq p'$ if p is an initial part of p'. Clearly, if G is acyclic, then the unfolding is a finite tree.

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 PSPACE-hard. For the case of acyclic DFAs and NFAs, respectively, we obtain completeness results for counting classes ($\mathsf{C_=L}$-completeness for acyclic DFAs and $\mathsf{C_=P}$-completeness for acyclic NFAs).

4.1. Upper bounds

Recall the definition of the generalized word $w(\mathcal{A})$ for a partitioned DFA from Section 2.5. The main result of this section is:

Theorem 18. *The following problem can be solved in polynomial time:*

INPUT: Two partitioned DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$.
QUESTION: $w(\mathcal{A}_1) \cong w(\mathcal{A}_2)$?

In Section 4.2–4.6 we prove Theorem 18. Section 4.2 will introduce some of the machinery from [3] concerning blocks. Blocks allow to condensate a generalized word to a coarser word (whose symbols are the blocks of the original word). In Section 4.3 we will formally introduce succinct regular expressions (expressions in form of dags) and in Section 4.4 we will argue that Heilbrunner's algorithm from [14] allows to transform a given partitioned DFA in polynomial time into an equivalent succinct (regular) expression. Hence, the remaining goal is to develop a polynomial time algorithm for checking whether two given succinct expressions represent isomorphic regular words. For the special case that these regular words consist of only one block (so called primitive regular words), this will be accomplished in Section 4.5. In this step, we will make use of algorithms for straight-line programs (succinctly represented finite words) [34]. Finally, in Section 4.6 we will present a polynomial time algorithm for checking whether two given succinct expressions represent isomorphic regular words.

4.2. Blocks and their combinatorics

In this section, we will introduce the crucial notion of a block, and we recall some of the results from [3] that we are using later.

Let $u = (L; \leq, \tau)$ be a generalized word over the possibly infinite alphabet Σ. An *interval* of u is an interval of the underlying linear order $(L; \leq)$. A *subword* or *factor* of u is an interval I of u together with the coloring τ restricted to I. Let $\Gamma \subseteq \Sigma$ be finite. A Γ-*uniform* subword of u is a subword that is isomorphic to Γ^η. A subword is *uniform* if it is Γ-uniform 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 or the predecessor of p exists, then it is contained in v as well. A successor-closed subword is *minimal* if it does not strictly contain another successor-closed subword. Following [3] we define:

Definition 19 (blocks). *Let u be a regular word. A* block *of 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.*[6] By [3] a word u is primitive if and only if it is of one of the following forms (where $x, z \in \Sigma^+$ and $y \in \Sigma^*$ are finite words): 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 Γ^η for some finite $\Gamma \subseteq \Sigma$. 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 a unique block $\mathsf{Bl}(p)$, which induces a regular (and hence primitive) word. Moreover we can order the blocks of u linearly by setting $\mathsf{Bl}(p) < \mathsf{Bl}(q)$ if and only if $p < q$. The order obtained that way is denoted $(\mathsf{Bl}(u); \leq)$. Then we extend the order $(\mathsf{Bl}(u); \leq)$ to a generalized word $\widehat{u}$ over $D(\Sigma)$ (here it is useful to allow infinite alphabets, since $D(\Sigma)$ is infinite), called the *skeleton* of u, by labeling each block with the corresponding isomorphic word in $D(\Sigma)$. Implicitly, it is shown in [3] that for every regular word u there exists a *finite* subset of $D(\Sigma)$ such that every block of u is isomorphic to a primitive word from that finite

[6] 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.

subset. Moreover, $\widehat{u}$ is again a regular word [3, Proposition 72]. Later it will be convenient to have the following renaming notion available. Let V be a finite alphabet, let $\varphi : V \to D(\Sigma)$ be an injective mapping and suppose that all blocks of a regular word u belong to the image of φ. The word v that has $(\mathsf{Bl}(u); \leq)$ as underlying order and each block B of u labeled with $\varphi^{-1}(B)$ is called the φ-skeleton of u. We will need the following result from [3]:

Proposition 20 ([3, Corollary 73]). *Let $u, v \in \mathsf{Reg}(\Sigma)$. Let V be a finite alphabet and let $\varphi : V \to D(\Sigma)$ be injective such that all blocks of u and v are in the image of φ. Then u and v are isomorphic if and only if the φ-skeletons of u and v are isomorphic.*

We will consider finite and infinite sequences, whose elements are regular words and where the underlying order type is either finite, ω, or $\overline{\omega}$. In the following, when writing $(u_i)_{i\in I}$, we assume that either $I = \{1, \ldots, n\} \neq \emptyset$ (i.e., $(u_i)_{i\in I}$ is the finite sequence $(u_1, \ldots, u_n)$) or $I = \{1, 2, 3, \ldots\}$ (i.e., $(u_i)_{i\in I}$ is the infinite sequence $(u_1, u_2, u_2, \ldots)$) or $I = \{\ldots, -2, -1, 0\}$ (i.e., $(u_i)_{i\in I}$ is the infinite sequence $(\ldots, u_{-2}, u_{-1}, u_0)$). The corresponding generalized word is $\prod_{i\in I} u_i$ (either $u_1 \cdots u_n$, or $u_1u_2u_3\cdots$, or $\cdots u_{-2}u_{-1}u_0$). We say that two sequences $(u_i)_{i\in I}$ and $(v_j)_{j\in J}$ are *equivalent*, if the generalized words $\prod_{i\in I} u_i$ and $\prod_{j\in J} v_j$ are isomorphic. We use commas to separate the successive u_i in the sequence $(u_i)_{i\in I}$ in order to avoid misinterpretations. For instance (a, a) viewed as a sequence over regular words has length two whereas (aa) has length 1. Of course, (a, a) and (aa) are equivalent sequences.

Definition 21. *Let $\bar{u} = (u_i)_{i\in I}$ be a sequence of regular words. We say that $\bar{u}$* merges *if there exists a block of $\prod_{i\in I} u_i$ that contains elements from two different u_i.*

Clearly, if a sequence $(u_i)_{i\in I}$ does not merge and J is an interval of I, then also $(u_i)_{i\in J}$ does not merge.

Example 22. *Clearly if u and v are finite words, then (u, v) merges. Also, $(\Gamma^\eta, \Gamma^\eta)$ and $(\Gamma^\eta, a, \Gamma^\eta)$ merge for every $\Gamma \subseteq \Sigma$ and $a \in \Gamma$ (in both cases, the sequence is equivalent to Γ^η). On the other hand, $([ab]^\eta, [ab]^\eta)$ does not merge. The reason is that the blocks of $[ab]^\eta$ are the copies of ab. More generally, if u is not primitive and X is a finite set of regular words, then $((X \cup \{u\})^\eta, (X \cup \{u\})^\eta)$ does not merge.*

Let us now restate several results from [3] that will be needed in the following.

Lemma 23 ([3, Corollary 32]). *A sequence $\bar{u}$ merges, if and only if there exists a factor (u_i, u_{i+1}) or (u_i, u_{i+1}, u_{i+2}) that merges.*

For the case of a sequence of primitive words, a complete description of merging sequences was given in [3]. Moreover, if a sequence of primitive words merges, then it can be simplified to a non-merging sequence of primitive words.

Lemma 24 ([3, Lemma 24]). *Let u, v, and w be primitive words.*

- *If (u, v) merges, then either u and v are Γ-uniform for some $\Gamma \subseteq \Sigma$ or u is right-closed and v is left-closed. In both cases, the regular word uv has a single block.*
- *If (u, v, w) merges, then either (u, v) merges, or (v, w) merges, or u, w are Γ-uniform and v is a singleton from Γ.*

Lemma 24 motivates the definition of the following rewriting system R over finite sequences over $D(\Sigma)$.

Definition 25 (rewriting system R). *The rewriting system R over the set $D(\Sigma)$ consists of the following rules:*

- *$(u_1, u_2, u_3) \to u$ if $u_1 = u_3 = u = \Gamma^\eta$ for some $\Gamma \subseteq \Sigma$ and $u_2 \in \Gamma$*
- *$(u_1, u_2) \to u$ if one of the following holds:*
  - *u_1 is right-closed, u_2 is left-closed and $u = u_1 u_2$*
  - *$u_1 = u_2 = u = \Gamma^\eta$ for some $\Gamma \subseteq \Sigma$.*

In the following, we will use some basic facts from rewriting theory, see e.g. [5] for further details. For sequences $\bar{x}$ and $\bar{y}$ over $\mathsf{Reg}(\Sigma)$, we write $\bar{x} \to_R \bar{y}$ if there exist a rewrite rule $\bar{u} \to u$ and an occurrence of the sequence $\bar{u}$ in $\bar{x}$ such that replacing that occurrence by u gives the sequence $\bar{y}$. Here, $\bar{x}$ and $\bar{y}$ may be infinite sequences. Moreover, those x_i of $\bar{x} = (x_i)_{i \in I}$ that are not primitive are left untouched in the rewrite step $\bar{x} \to_R \bar{y}$. Clearly, $\bar{x} \to_R \bar{y}$ implies that the sequences $\bar{x}$ and $\bar{y}$ are equivalent. A (possibly infinite) sequence $\bar{u}$ is irreducible with respect to R if there does not exist a sequence

$\bar{v}$ with $\bar{u} \to_R \bar{v}$. Clearly, on infinite sequences, R cannot be *terminating* (e.g., $(a^\eta, a^\eta, a^\eta \ldots) \to_R (a^\eta, a^\eta, a^\eta \ldots)$ is a loop). On the other hand, R is trivially terminating on finite sequences, since it is length-reducing. Moreover, by analyzing overlapping left-hand sides of R, one can easily show:

Lemma 26. *The rewriting system R is strongly confluent (on finite and infinite sequences), i.e., for all $\bar{u}$, $\bar{v}$, $\bar{w}$ such that $\bar{u} \to_R \bar{v}$ and $\bar{u} \to_R \bar{w}$ there exists $\bar{x}$ such that ($\bar{v} = \bar{x}$ or $\bar{v} \to_R \bar{x}$) and ($\bar{w} = \bar{x}$ or $\bar{w} \to_R \bar{x}$).*

Proof. The only possible overlappings of left-hand sides of R are the following:

- $\bar{u} = (\Gamma^\eta, a, \Gamma^\eta, a, \Gamma^\eta)$ for $\Gamma \subseteq \Sigma$ and $a \in \Gamma$: We obtain $(\Gamma^\eta, a, \Gamma^\eta)$ regardless of whether we apply the rule $(\Gamma^\eta, a, \Gamma^\eta) \to \Gamma^\eta$ to the first or the second occurrence, respectively, of the left-hand side $(\Gamma^\eta, a, \Gamma^\eta)$ in $\bar{u}$.

- $\bar{u} = (\Gamma^\eta, \Gamma^\eta, a, \Gamma^\eta)$ for $\Gamma \subseteq \Sigma$ and $a \in \Gamma$: We have $\bar{u} \to_R (\Gamma^\eta, a, \Gamma^\eta) \to_R \Gamma^\eta$ and $\bar{u} \to_R (\Gamma^\eta, \Gamma^\eta) \to_R \Gamma^\eta$.

- $\bar{u} = (\Gamma^\eta, a, \Gamma^\eta, \Gamma^\eta)$ for $\Gamma \subseteq \Sigma$ and $a \in \Gamma$: Analogously to the previous case.

- $\bar{u} = (\Gamma^\eta, \Gamma^\eta, \Gamma^\eta)$ for $\Gamma \subseteq \Sigma$: We obtain $(\Gamma^\eta, \Gamma^\eta)$ regardless of whether we apply the rule $(\Gamma^\eta, \Gamma^\eta) \to \Gamma^\eta$ to the first or the second occurrence, respectively, of the left-hand side $(\Gamma^\eta, \Gamma^\eta)$ in $\bar{u}$.

- $\bar{u} = (u_1, u_2, u_3)$, where $u_2 \in \Sigma^*$ is a finite word, u_1 is right-closed, and u_3 is left-closed: We have $\bar{u} \to_R (u_1 u_2, u_3) \to_R u_1 u_2 u_3$ and $\bar{u} \to_R (u_1, u_2 u_3) \to_R u_1 u_2 u_3$. □

By a simple fact from rewriting theory, it follows that R is also *confluent*, i.e., for all $\bar{u}$, $\bar{v}$, $\bar{w}$ such that $\bar{u} \to_R^* \bar{v}$ and $\bar{u} \to_R^* \bar{w}$ there exists $\bar{x}$ such that $\bar{v} \to_R^* \bar{x}$ and $\bar{w} \to_R^* \bar{x}$. Termination (on finite sequences) and confluence imply that R produces unique normal forms for finite sequences, i.e., for every finite sequence $\bar{u}$ there exists a unique finite sequence $\bar{v}$ such that $\bar{u} \to_R^* \bar{v}$ and $\bar{v}$ is irreducible with respect to R. This $\bar{v}$ is called the *irreducible normal form* of $\bar{u}$. From Lemma 23 and 24 we immediately get:

Lemma 27. *Let $\bar{u}$ be a sequence of primitive words. Then $\bar{u}$ does not merge if and only if $\bar{u}$ is irreducible with respect to R.*

We also have to verify that a sequence $\bar{u}$ over $\mathsf{Reg}(\Sigma)$ containing non-primitive words does not merge. Note that a regular word needs not have a first or last block. For instance, $(a^\omega)^\omega$ has a first block but no last block, whereas $(a^\omega)^{\overline{\omega}}(a^\omega)^\omega$ and $[aa]^\eta$ neither have a first block nor a last block.

Lemma 28 ([3, Corollary 30 and 31]). *Let u, v, and w be regular words.*

(1) If u has no last block or v has no first block then (u, v) does not merge (see statement (1) from [3, Corollary 30]).

(2) If (u, v) and (v, w) do not merge, and u has no last block or w has no first block or v has more than one block, then (u, v, w) does not merge (see statement (1) from [3, Corollary 31]).

(3) If u has a last block b, w has a first block c and v has a single block, then (u, v, w) merges if and only if (b, v, c) merges (see statement (2) from [3, Corollary 31]).

We use the following definition.

Definition 29 (good sequences and semi-good sequences). *The sequence $\bar{u} = (u_i)_{i\in I}$ is* good *if the following conditions hold:*

(1) $\bar{u}$ is irreducible with respect to R.

(2) For all $i \in I$ we have:

(a) If u_i is not primitive and has a first block, then ($i - 1 \in I$, u_{i-1} is uniform, and (u_{i-1}, u_i) does not merge) or ($i - 1, i - 2 \in I$, u_{i-1} and u_{i-2} are primitive, and (u_{i-2}, u_{i-1}, u_i) does not merge).

(b) If u_i is not primitive and has a last block, then ($i + 1 \in I$, u_{i+1} is uniform, and (u_i, u_{i+1}) does not merge) or ($i + 1, i + 2 \in I$, u_{i+1} and u_{i+2} are primitive, and (u_i, u_{i+1}, u_{i+2}) does not merge).

If only (2) holds, then $\bar{u}$ is said to be semi-good.

Lemma 30. *If $\bar{u}$ merges, then $\bar{u}$ is not good.*

Proof. Assume that $\bar{u}$ merges. By Lemma 23 one of the following cases holds:

Case 1. $\bar{u}$ contains a factor (u_i, u_{i+1}) that merges. If u_i and u_{i+1} are both primitive, then $\bar{u}$ is not irreducible by Lemma 27. Hence $\bar{u}$ is not good. Now assume that u_i or u_{i+1} is not primitive. W.l.o.g. assume that u_i is not primitive; the other case is symmetric. Since (u_i, u_{i+1}) merges, Lemma 28(1) implies that u_i has a last block. Since (u_i, u_{i+1}) merges, condition (2b) from Definition 29 implies that $\bar{u}$ is not good.

Case 2. $\bar{u}$ contains a factor (u_i, u_{i+1}, u_{i+2}) that merges but neither (u_i, u_{i+1}) nor (u_{i+1}, u_{i+2}) merges. If u_i, u_{i+1}, and u_{i+2} are primitive, then $\bar{u}$ is not irreducible by Lemma 27 and hence not good. So, assume that u_i, u_{i+1}, or u_{i+2} is not primitive. The case that u_{i+2} is not primitive is symmetric to the case that u_i is not primitive. Hence, it suffices to consider the following two subcases:

Case 2a. u_i is not primitive. Since (u_i, u_{i+1}, u_{i+2}) merges but (u_i, u_{i+1}) and (u_{i+1}, u_{i+2}) do not merge, Lemma 28(2) implies that u_i has a last block b_i and u_{i+2} has a first block b_{i+2}. If u_{i+1} is not uniform, then, since (u_i, u_{i+1}, u_{i+2}) merges, condition (2b) from Definition 29 implies that $\bar{u}$ is not good. Now, assume that u_{i+1} is uniform and hence has a single block. Lemma 28(3) implies that (b_i, u_{i+1}, b_{i+2}) merges. Since (u_i, u_{i+1}) and (u_{i+1}, u_{i+2}) do not merge, also (b_i, u_{i+1}) and (u_{i+1}, b_{i+2}) do not merge. It follows (from the form of our rewriting system R) that $b_i = b_{i+2}$ is uniform and u_{i+1} is a singleton word. But we have already shown that u_{i+1} is uniform, which is a contradiction.

Case 2b. u_{i+1} is not primitive. Then u_{i+1} has more than one block and Lemma 28(2) directly implies that (u_i, u_{i+1}, u_{i+2}) does not merge, which is again a contradiction. □

Lemma 31. *If $\bar{u}$ is semi-good and $\bar{u} \to_R \bar{v}$, then $\bar{v}$ is semi-good as well.*

Proof. Assume that $\bar{u} = (u_i)_{i \in I}$ is semi-good and $\bar{u} \to_R \bar{v}$. We have to show that $\bar{v} = (v_j)_{j \in J}$ is semi-good. For this, consider an $j \in J$ such that v_j is not primitive. Since the system R does not introduce non-primitive words, v_j must have been already present in $\bar{u}$. Let $i \in I$ be the position in $\bar{u}$ that corresponds to position j in $\bar{v}$. Hence, $u_i = v_j$. By symmetry it suffices to show that condition (2a) from Definition 29 holds for $j \in J$. The case that

$u_i = v_j$ has no first block is clear. So, assume that u_i has a first block. Since $\bar{u}$ is semi-good, we can distinguish the following two cases.

Case 1. $i-1 \in I$, u_{i-1} is uniform, and (u_{i-1}, u_i) does not merge. From the form of the rewrite rules, it follows that $v_{j-1} = u_{i-1}$. Hence, v_{j-1} is uniform, and $(v_{j-1}, v_j) = (u_{i-1}, u_i)$ does not merge. Thus, we have shown condition (2a) from Definition 29 for j.

Case 2. $i-1, i-2 \in I$, u_{i-2}, u_{i-1} are primitive, and (u_{i-2}, u_{i-1}, u_i) does not merge. We make a case distinction on the position, where the rewrite rule is applied.

Case 2a. $i-3 \in I$ and in the rewrite step $\bar{u} \to_R \bar{v}$, $(u_{i-3}, u_{i-2}, u_{i-1})$ is replaced by $u \in D(\Sigma)$. Thus, $u_{i-3} = u_{i-1} = u$ is uniform. Hence, $v_{j-1} = u$ is uniform. Moreover, $(v_{j-1}, v_j) = (u_{i-1}, u_i)$ does not merge.

Case 2b. $i-4 \in I$ and in the rewrite step $\bar{u} \to_R \bar{v}$, $(u_{i-4}, u_{i-3}, u_{i-2})$ is replaced by $u \in D(\Sigma)$. Thus, $u_{i-4} = u_{i-2} = u$ is uniform, $v_{j-2} = u = u_{i-2}$, and $u_{i-1} = v_{j-1}$. It follows that v_{j-2} and v_{j-1} are primitive, and that $(v_{j-2}, v_{j-1}, v_j) = (u_{i-2}, u_{i-1}, u_i)$ does not merge.

Case 2c. In the rewrite step $\bar{u} \to_R \bar{v}$, (u_{i-2}, u_{i-1}) is replaced by $u \in D(\Sigma)$. Then, (u_{i-2}, u_{i-1}) merges. But this contradicts the assumption that (u_{i-2}, u_{i-1}, u_i) does not merge.

Case 2d. $i-3 \in I$ and in the rewrite step $\bar{u} \to_R \bar{v}$, (u_{i-3}, u_{i-2}) is replaced by $u \in D(\Sigma)$. If $u_{i-3} = u_{i-2} = u$ is uniform, then $v_{j-2} = u_{i-2}$ and $v_{j-1} = u_{i-1}$ are primitive and $(v_{j-2}, v_{j-1}, v_j) = (u_{i-2}, u_{i-1}, u_i)$ does not merge. Finally, assume that u_{i-3} is right-closed, u_{i-2} is left-closed and $v_{j-2} = u = u_{i-3}u_{i-2}$. We have $v_{j-1} = u_{i-1}$. Thus v_{j-1} and v_{j-2} are primitive. It remains to show that $(v_{j-2}, v_{j-1}, v_j) = (u_{i-3}u_{i-2}, u_{i-1}, u_i)$ does not merge. We know that (u_{i-1}, u_i) does not merge (since (u_{i-2}, u_{i-1}, u_i) does not merge). Assume that $(u_{i-3}u_{i-2}, u_{i-1})$ merges. Then (since $u_{i-3}u_{i-2}$ is primitive and scattered and u_{i-1} is primitive) $u_{i-3}u_{i-2}$ must be right-closed and u_{i-1} must be left-closed. But then, $u_{i-2} \neq \varepsilon$ is right-closed as well and (u_{i-2}, u_{i-1}) merges. This is a contradiction. Hence, $(u_{i-3}u_{i-2}, u_{i-1})$ does not merge. Let b_i be the first block of u_i; we know that it exists. If $(u_{i-3}u_{i-2}, u_{i-1}, u_i)$ merges, then by Lemma 28(3) $(u_{i-3}u_{i-2}, u_{i-1}, b_i)$ merges. Since neither $(u_{i-3}u_{i-2}, u_{i-1})$ nor (u_{i-1}, b_i) merges, $u_{i-3}u_{i-2}$ and b_i must be uniform. But we know that $u_{i-3}u_{i-2}$ is scattered, which leads again to a contradiction. Thus, indeed $(u_{i-3}u_{i-2}, u_{i-1}, u_i)$ does not merge.

If the rewrite rule is applied at a position different from those considered in Cases 2a–2d, then we have $(v_{j-2}, v_{j-1}, v_j) = (u_{i-2}, u_{i-1}, u_i)$. Since (u_{i-2}, u_{i-1}, u_i) fulfills condition (2a) from Definition 29, so does (v_{j-2}, v_{j-1}, v_j). This concludes the proof of the lemma. □

Lemma 31 implies that from a given finite semi-good sequence $\bar{u}$ we can compute an equivalent good sequence by computing the (unique) irreducible normal form of $\bar{u}$.

4.3. Expressions and succinct expressions

Regular words can be naturally described by expressions using the operations of concatenation, ω-power, $\overline{\omega}$-power, and shuffle. Formally, the set $T(V, \Sigma)$ of *expressions* over V and Σ is inductively defined as follows:

(a) $V \cup \Sigma \subseteq T(V, \Sigma)$

(b) If $\alpha_1, \dots, \alpha_n \in T(V, \Sigma)$ $(n \geq 1)$, then $\alpha_1 \cdots \alpha_n \in T(V, \Sigma)$.

(c) If $\alpha \in T(V, \Sigma)$, then $\alpha^\omega \in T(V, \Sigma)$ and $\alpha^{\overline{\omega}} \in T(V, \Sigma)$.

(d) If $\alpha_1, \dots, \alpha_n \in T(V, \Sigma)$ $(n \geq 1)$, then $[\alpha_1, \dots, \alpha_n]^\eta \in T(V, \Sigma)$.

A mapping $f : V \to \mathsf{Reg}(\Sigma)$ will be extended homomorphically to a mapping $f : T(V, \Sigma) \to \mathsf{Reg}(\Sigma)$ inductively as follows, where $\alpha, \alpha_1, \dots, \alpha_n \in T(V, \Sigma)$:

- $f(a) = a$ for $a \in \Sigma$
- $f(\alpha_1 \cdots \alpha_n) = f(\alpha_1) \cdots f(\alpha_n)$
- $f(\alpha^\omega) = f(\alpha)^\omega$
- $f(\alpha^{\overline{\omega}}) = f(\alpha)^{\overline{\omega}}$
- $f([\alpha_1, \dots, \alpha_n]^\eta) = [f(\alpha_1), \dots, f(\alpha_n)]^\eta$

For $\alpha \in T(V, \Sigma)$ we define the size $|\alpha| \in \mathbb{N}$ inductively as follows:

- $|\alpha| = 1$ for $\alpha \in V \cup \Sigma$
- $|\alpha_1 \cdots \alpha_n| = |\alpha_1| + \cdots + |\alpha_n|$
- $|\alpha^\omega| = |\alpha^{\overline{\omega}}| = |\alpha| + 1$

- $|[\alpha_1, \dots, \alpha_n]^\eta| = |\alpha_1| + \cdots + |\alpha_n| + 1$

A *succinct expression system (SES)* is a triple $\mathbb{A} = (V, \Sigma, \mathsf{rhs})$ such that:

- V (the set of variables) and Σ (the terminal alphabet) are disjoint finite alphabets.
- rhs (for right-hand side) is a mapping from V to $T(V, \Sigma)$ such that the relation $\{(Y, X) \in V \times V \mid Y \text{ occurs in } \mathsf{rhs}(X)\}$ is acyclic. The reflex transitive closure of this relation is called the *hierarchical order* of $\mathbb{A}$ and denoted by $\preceq_{\mathbb{A}}$.

The property for rhs ensures that there exists a unique mapping $\mathsf{val}_{\mathbb{A}} : V \to \mathsf{Reg}(\Sigma)$ such that $\mathsf{val}_{\mathbb{A}}(X) = \mathsf{val}_{\mathbb{A}}(\mathsf{rhs}(X))$ for all $X \in V$. If $\mathbb{A}$ is clear from the context, we will simply write $\mathsf{val}(X)$.

In the following a quadruple $\mathbb{A} = (V, \Sigma, \mathsf{rhs}, S)$ where $(V, \Sigma, \mathsf{rhs})$ is as above and $S \in V$ (i.e., an SES with a distinguished start variable S) we will be called a *succinct expression.* In this case let us set $\mathsf{val}(\mathbb{A}) = \mathsf{val}_{\mathbb{A}}(S)$. A succinct expression may be also seen as a dag (directed acyclic graph), whose unfolding is an expression in the above sense.

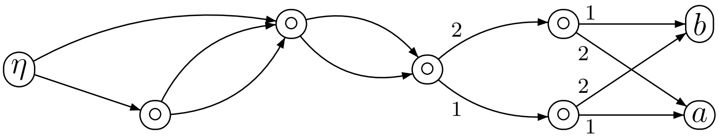
Example 32. *Consider the succinct expression*

$$\mathbb{A} = (\{X_1, X_2, X_3, X_4, X_5\}, \{a, b\}, \mathsf{rhs}, X_1)$$

with

$$\begin{array}{lll} \mathsf{rhs}(X_1) = [X_2, X_3]^\eta & \mathsf{rhs}(X_2) = X_3 X_3 & \mathsf{rhs}(X_3) = X_4 X_4 \\ \mathsf{rhs}(X_4) = X_5 X_6 & \mathsf{rhs}(X_5) = ab & \mathsf{rhs}(X_6) = ba. \end{array}$$

We have $\mathsf{val}(\mathbb{A}) = [abbaabba, abbaabbaabbaabba]^\eta$. *The corresponding dag looks as follows:*



Nodes labelled with $\circ$ *compute the concatenation of their successor nodes. In case the order of the successor nodes matters, we specify it by edge labels.*

For an SES $\mathbb{A}$ we define

$$|\mathbb{A}| = \sum_{X \in V} |\mathsf{rhs}(X)|.$$

An SES $\mathbb{A} = (V, \Sigma, \mathsf{rhs})$ is in *normal form* if all right-hand sides are in $(V \cup \Sigma)^+$ or of the form $Y^\omega, Y^{\overline{\omega}}, [Y_1, \ldots, Y_n]^\eta$ for some $Y, Y_1, \ldots, Y_n \in V \cup \Sigma$. Clearly, from an SES $\mathbb{A} = (V, \Sigma, \mathsf{rhs})$ we can construct in polynomial time an SES $\mathbb{B} = (V', \Sigma, \mathsf{rhs}')$ in normal form such that $V \subseteq V'$ and $\mathsf{val}_{\mathbb{A}}(X) = \mathsf{val}_{\mathbb{B}}(X)$ for all $X \in V$. In the following we will only consider SESs in normal form.

For an SES $\mathbb{A}$ in normal form, we define $\mathsf{depth}_{\mathbb{A}}(X)$ and $\omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(X)$ for $X \in V$ inductively as follows (below, we set $\mathsf{depth}_{\mathbb{A}}(a) = \omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(a) = 0$ for $a \in \Sigma$):

- If $\mathsf{rhs}(X) = Y_1 \cdots Y_n$ ($n \geq 1$, $Y_1, \ldots, Y_n \in \Sigma \cup V$), then

$$\begin{aligned} \mathsf{depth}_{\mathbb{A}}(X) &= \max(\mathsf{depth}_{\mathbb{A}}(Y_1), \ldots, \mathsf{depth}_{\mathbb{A}}(Y_n)) + 1, \\ \omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(X) &= \max(\omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(Y_1), \ldots, \omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(Y_n)). \end{aligned}$$

- If $\mathsf{rhs}(X) = Y^\omega$ or $\mathsf{rhs}(X) = Y^{\overline{\omega}}$, then

$$\begin{aligned} \mathsf{depth}_{\mathbb{A}}(X) &= \mathsf{depth}_{\mathbb{A}}(Y) + 1, \\ \omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(X) &= \omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(Y) + 1. \end{aligned}$$

- If $\mathsf{rhs}(X) = [Y_1, \ldots, Y_n]^\eta$, then

$$\begin{aligned} \mathsf{depth}_{\mathbb{A}}(X) &= \max(\mathsf{depth}_{\mathbb{A}}(Y_1), \ldots, \mathsf{depth}_{\mathbb{A}}(Y_n)) + 1, \\ \omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(X) &= \max(\omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(Y_1), \ldots, \omega\eta\text{-}\mathsf{depth}_{\mathbb{A}}(Y_n)) + 1. \end{aligned}$$

Straight-line programs.. A succinct expression, where all right-hand sides belong to $(V \cup \Sigma)^+$ is called a *straight-line program (SLP)* [26, 32, 34]. In this case, $\mathsf{val}(\mathbb{A})$ is a finite non-empty word. An SLP $\mathbb{A}$ can be viewed as a succinct representation of the word $\mathsf{val}(\mathbb{A})$. More precisely, the length of $\mathsf{val}(\mathbb{A})$ may be exponential in $|\mathbb{A}|$. We will make heavy use of the fact that certain algorithmic problems on SLP-encoded finite words can be solved in polynomial time. More precisely, we use the following results:

Remark 33. *There exist polynomial time algorithms for the following problems (the proofs for (a), (b), and (c) are straightforward):*

(a) Given an SLP $\mathbb{A}$, calculate $|\mathsf{val}(\mathbb{A})|$.

(b) *Given an SLP $\mathbb{A}$ and a number $k \in \mathbb{N}$ (coded in binary) we can produce an SLP $\mathbb{B}$ of size $|\mathbb{A}| + O(\log k)$ such that* $\mathsf{val}(\mathbb{B}) = \mathsf{val}(\mathbb{A})^k$.

(c) *Given an SLP $\mathbb{A}$ and numbers $1 \leq i \leq j \leq |\mathsf{val}(\mathbb{A})|$, compute an SLP $\mathbb{B}$ with* $\mathsf{val}(\mathbb{B}) = \mathsf{val}(\mathbb{A})[i : j]$. *Here $w[i : j] = a_i \ldots a_j$ for a finite word $w = a_1 \ldots a_n$.*

(d) *Given SLPs $\mathbb{A}$ and $\mathbb{B}$ decide whether* $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$ *[15, 28, 31].*

(e) *Given SLPs $\mathbb{A}$ and $\mathbb{B}$ decide whether* $\mathsf{val}(\mathbb{A})$ *is a factor of* $\mathsf{val}(\mathbb{B})$, *i.e., there exist finite words u and v such that* $u\,\mathsf{val}(\mathbb{A})v = \mathsf{val}(\mathbb{B})$ *[12, 17, 24, 29].*

4.4. Heilbrunner's algorithm

Theorem 34. *From a given partitioned DFA $\mathcal{A}$, we can compute in polynomial time a succinct expression $\mathbb{A}$ such that $w(\mathcal{A}) \cong$* $\mathsf{val}(\mathbb{A})$.

Proof. There is nothing new about the proof. We just have to follow Heilbrunner's algorithm [14] carefully. Let $\mathcal{A} = (Q, \Gamma, \delta, q_0, (F_a)_{a\in\Sigma})$ be a partitioned DFA and let $F = \bigcup_{a\in\Sigma} F_a$. We can assume that every state in F is a dead end, i.e., does not have outgoing transitions. For this, take a new symbol \$, as well as a copy q' together with the transition $(q, \$, q')$ for every final state $q \in F$. We set $F'_a = \{q' \mid q \in F_a\}$ and let \$ be the smallest symbol in $\Gamma \cup \{\$\}$. The resulting partitioned DFA produces the same generalized word as $\mathcal{A}$.

So, assume that every state in F is a dead end. W.l.o.g. we can also assume that $\mathcal{A}$ is coaccessible. The variables of the succinct expression $\mathbb{A}$ will be the states of $\mathcal{A}$. Consider a state $p \in Q$ and let (p, a_i, q_i) $(1 \leq i \leq k)$ be all outgoing transitions for p, where $a_1 < a_2 < \cdots < a_k$. Let us define $\mathsf{out}(p) = q_1 q_2 \cdots q_k$. Next, consider the graph with node set Q and an edge from $p \in Q$ to $q \in Q$ if there is a transition from p to q. We partition this graph into its strongly connected components (SCCs). An SCC C is smaller than an SCC D if there exists a path from a state in C to a state in D; this defines a partial order on the set of SCCs. We process all SCCs starting with the maximal ones. When processing an SCC C, we define $\mathsf{rhs}_{\mathbb{A}}(p)$ for each state $p \in C$. So let us consider a maximal SCC C which has not been processed so far. We distinguish three cases:

Case 1. C is a singleton set $\{p\}$ with $p \in F_a$. Then we set $\mathsf{rhs}_{\mathbb{A}}(p) = a$.

Case 2. $C = \{p\}$ is a singleton set with $p \notin F$, and p does not occur in $\mathsf{out}(p)$. We set $\mathsf{rhs}_{\mathbb{A}}(p) = \mathsf{out}(p)$. Note that $\mathsf{out}(p)$ only contains states from larger SCCs (which have already been processed) and that $\mathsf{out}(p) \neq \varepsilon$, since $p \notin F$ and $\mathcal{A}$ is coaccessible.

Case 3. Neither Case 1 or Case 2 applies, which means that either $|C| \geq 2$ or $C = \{p\}$ and p occurs in $\mathsf{out}(p)$. Then every word $\mathsf{out}(p)$ $(p \in C)$ contains at least one occurrence of a state from C. Hence $\mathsf{out}(p)$ can be factored as $\mathsf{out}(p) = u_p x_p v_p$, where u_p and v_p do not contain occurrences of states from the SCC C (i.e., all states occurring in u_p and v_p belong to larger SCCs), and x_p starts and ends with a state from C (x_p might consist of a single state from C). Define functions $\ell : C \to C$ and $r : C \to C$ as follows: $\ell(p)$ (resp. $r(p)$) is the first (resp. last) state of the word x_p. Then, for every $p \in C$, the sequences $p, \ell(p), \ell^2(p), \ldots$ and $p, r(p), r^2(p), \ldots$ become periodic after at most $|C|$ steps. We now define regular expressions ℓ_p and r_p as follows: Let $p_0, p_1, \ldots, p_m$ and $q_0, q_1, \ldots, q_m$ be shortest sequences such that $p_0 = q_0 = p$, $p_{i+1} = \ell(p_i)$, $q_{i+1} = r(q_i)$, and $\ell(p_m) \in \{p_0, p_1, \ldots, p_m\}$, $r(q_n) \in \{q_0, q_1, \ldots, q_n\}$. Assume that $\ell(p_m) = p_{m'}$ and $r(q_n) = q_{n'}$ for $0 \leq m' \leq m$, $0 \leq n' \leq n$. Then, we define

$$\begin{aligned} \ell_p &= (u_{p_0} \cdots u_{p_{m'-1}})(u_{p_{m'}} \cdots u_{p_m})^{\omega}, \\ r_p &= (v_{q_n} \cdots v_{q_{n'}})^{\overline{\omega}}(v_{q_{n'-1}} \cdots v_{q_0}). \end{aligned}$$

Next, let T be the set of all regular expressions of the form $\ell_s y r_t$ $(s, t \in C)$ such that some word $\mathsf{out}(p)$ $(p \in C)$ contains a factor syt, where the word y does not contain a state from C. Then we finally set $\mathsf{rhs}_{\mathbb{A}}(p) = \ell_p[T]^{\eta} r_p$ for all $p \in C$. This concludes the processing step for the SCC C. Since C is not maximal (since $\mathcal{A}$ is coaccessible), there exist edges from C to a larger SCC. This implies that $\mathsf{rhs}_{\mathbb{A}}(p)$ is not empty. By [14], for every state $p \in Q$ we have $w(Q, \Gamma, \delta, p, (F_a)_{a \in \Sigma}) \cong \mathsf{val}_{\mathbb{A}}(p)$. □

By Theorem 34, it suffices to prove the following result in order to prove Theorem 18.

Theorem 35. *The following problem can be solved in polynomial time:*

INPUT: Two succinct expressions $\mathbb{A}_1$ *and* $\mathbb{A}_2$.
QUESTION: $\mathsf{val}(\mathbb{A}_1) \cong \mathsf{val}(\mathbb{A}_2)$*?*

In the next section, we will prove this result for the special case that both $\mathsf{val}(\mathbb{A}_1)$ and $\mathsf{val}(\mathbb{A}_2)$ are primitive.

4.5. A polynomial time equivalence test for succinct primitive expressions

By Theorem 34, the remaining goal is to test in polynomial time, whether two succinct expressions represent isomorphic regular words. In a first step, we accomplish this for succinct expressions that represent primitive words. In the following, Σ will always refer to a *finite* alphabet. Let us first show that we can decide in polynomial time whether a succinct expression represents a primitive word.

Lemma 36. *Given a succinct expression $\mathbb{A}$, we can decide in polynomial time whether $\mathsf{val}(\mathbb{A})$ is a primitive word. If it is, we can compute in polynomial time SLPs $\mathbb{B}, \mathbb{C}, \mathbb{D}$ and $\Gamma \subseteq \Sigma$ (here, we should allow also the empty word for $\mathsf{val}(\mathbb{C})$) such that $\mathsf{val}(\mathbb{A})$ is equal to $\mathsf{val}(\mathbb{B})$ or $\mathsf{val}(\mathbb{C})\mathsf{val}(\mathbb{D})^\omega$ or $\mathsf{val}(\mathbb{B})^{\overline{\omega}}\mathsf{val}(\mathbb{C})$ or $\mathsf{val}(\mathbb{B})^{\overline{\omega}}\mathsf{val}(\mathbb{C})\mathsf{val}(\mathbb{D})^\omega$ or Γ^η, and we can decide which case applies.*

Proof. We proceed along the hierarchical order of $\mathbb{A}$ and compute for each variable A of $\mathbb{A}$ whether $\mathsf{val}(A)$ is of one of the following forms ($u, w \in \Sigma^+, v \in \Sigma^*$, $\Gamma \subseteq \Sigma$, $a, b \in \Gamma$): u, $u^{\overline{\omega}}v$, vw^ω, $u^{\overline{\omega}}vw^\omega$, Γ^η, $a\Gamma^\eta$, $\Gamma^\eta b$, $a\Gamma^\eta b$. Moreover, SLPs for the finite words u, v, and w can computed simultaneously. Observe that from $\mathsf{rhs}(A)$ and the information already computed we can easily obtain whether $\mathsf{val}(A)$ is of such a form and in this case of which form. The following identities have to be used for shuffles ($\Gamma \subseteq \Sigma$, $n \geq 0$, $m \geq 1$, $a, a_1, \ldots, a_n \in \Gamma$, and every u_i ($1 \leq i \leq m$) has one of the forms Γ^η, $c\Gamma^\eta$, $\Gamma^\eta c$, $c\Gamma^\eta d$ with $c, d \in \Gamma$)

$$[a_1, \ldots, a_n, u_1, \ldots, u_m]^\eta \cong \Gamma^\eta \tag{8}$$

$$\Gamma^\eta\Gamma^\eta \cong \Gamma^\eta a\Gamma^\eta \cong (\Gamma^\eta)^\omega \cong (\Gamma^\eta)^{\overline{\omega}} \cong (\Gamma^\eta a)^\omega \cong (a\Gamma^\eta)^{\overline{\omega}} \cong \Gamma^\eta \tag{9}$$

$$(a\Gamma^\eta)^\omega \cong a\Gamma^\eta \tag{10}$$

$$(\Gamma^\eta a)^{\overline{\omega}} \cong \Gamma^\eta a \tag{11}$$

The identities in (8) and (9) are axioms for regular expressions in [3], and (10) and (11) can be easily deduced from the axioms. Let us also note that in (8) it is crucial that $m > 0$. This allows to only require $\{a_1, \ldots, a_n\} \subseteq \Gamma$ instead of $\{a_1, \ldots, a_n\} = \Gamma$.

Now $\mathsf{val}(\mathbb{A})$ is primitive if and only if $\mathsf{val}(S)$ is of one of the following forms ($u, w \in \Sigma^+, v \in \Sigma^*$, $\Gamma \subseteq \Sigma$): u, $u^{\overline{\omega}}v$, vw^ω, $u^{\overline{\omega}}vw^\omega$, Γ^η. □

For our polynomial time equivalence test for succinct expressions that represent primitive words, we need the following technical lemma.

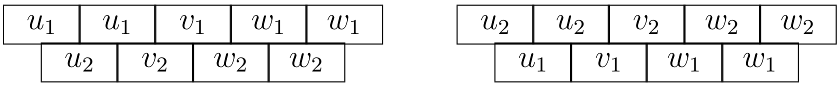


Figure 9:

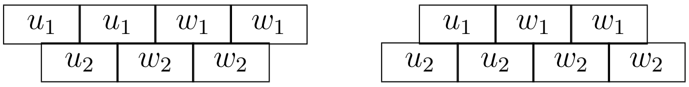


Figure 10:

Lemma 37. *Let u_i, v_i, w_i ($i \in \{1,2\}$) be finite words such that $|u_1| = |u_2| = |v_1| = |v_2| = |w_1| = |w_2| > 0$. Then $u_1^{\overline{\omega}} v_1 w_1^{\omega} = u_2^{\overline{\omega}} v_2 w_2^{\omega}$ if and only if one of the following conditions holds:*

- *$u_2 v_2 w_2^2$ is a factor of $u_1^2 v_1 w_1^2$.*
- *$u_1 v_1 w_1^2$ is a factor of $u_2^2 v_2 w_2^2$.*
- *$v_1 = w_1$, $u_2 = v_2$, and $u_2 w_2^2$ is a factor of $u_1^2 w_1^2$.*
- *$u_1 = v_1$, $v_2 = w_2$, and $u_1 w_1^2$ is a factor of $u_2^2 w_2^2$.*

Proof. The four conditions from the lemma are shown in Figures 9 and 10. It is straightforward to show that any of these four situations implies $u_1^{\overline{\omega}} v_1 w_1^{\omega} = u_2^{\overline{\omega}} v_2 w_2^{\omega}$. For instance, if the left situation in Figure 9 occurs, then there exist words x, y, x', y' such that $u_1 = xy$, $u_2 = yx$, $w_1 = x'y'$, $w_2 = y'x'$ and $v_2 w_2 = y v_1 x'$. Hence,

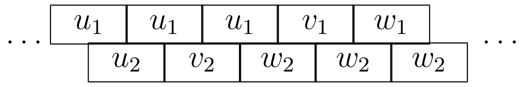
$$u_1^{\overline{\omega}} v_1 w_1^{\omega} = (xy)^{\overline{\omega}} v_1 (x'y')^{\omega} = (yx)^{\overline{\omega}} y v_1 x' (y'x')^{\omega} = u_2^{\overline{\omega}} v_2 w_2 w_2^{\omega} = u_2^{\overline{\omega}} v_2 w_2^{\omega}.$$

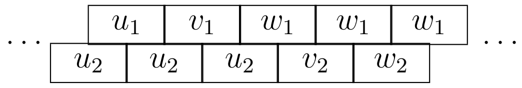
Let us now assume that $u_1^{\overline{\omega}} v_1 w_1^{\omega} = u_2^{\overline{\omega}} v_2 w_2^{\omega}$. We distinguish the following cases:

Case 1. The occurrence of v_1 in $u_1^{\overline{\omega}} v_1 w_1^{\omega}$ overlaps the occurrence of v_2 in $u_2^{\overline{\omega}} v_2 w_2^{\omega}$. Then, either $u_2 v_2 w_2^2$ is a factor of $u_1^2 v_1 w_1^2$ (if v_2 starts before v_1) or $u_1 v_1 w_1^2$ is a factor of $u_2^2 v_2 w_2^2$ (if v_1 starts before v_2), see Figure 9.

Case 2. The occurrence of v_1 in $u_1^{\overline{\omega}} v_1 w_1^{\omega}$ does not overlap the occurrence of v_2 in $u_2^{\overline{\omega}} v_2 w_2^{\omega}$.

Case 2.1. The occurrence of $u_1v_1w_1$ in $u_1^{\overline{\omega}}v_1w_1^{\omega}$ overlaps the occurrence of v_2 in $u_2^{\overline{\omega}}v_2w_2^{\omega}$. Then, one of the following two situations occurs:





In the first situation, we obtain $v_1 = w_1$ (since v_1w_1 is a factor of w_2^3) and $u_2 = v_2$ (since u_2v_2 is a factor of u_1^3). Hence, we get the left situation shown in Figure 10, i.e., $u_2w_2^2$ is a factor of $u_1^2w_1^2$. In the second situation, we obtain $u_1 = v_1$ (since u_1v_1 is a factor of u_2^3) and $v_2 = w_2$ (since v_2w_2 is a factor of w_1^3). Hence, we get the right situation shown in Figure 10, i.e., $u_1w_1^2$ is a factor of $u_2^2w_2^2$.

Case 2.2. The occurrence of $u_1v_1w_1$ in $u_1^{\overline{\omega}}v_1w_1^{\omega}$ does not overlap the occurrence of v_2 in $u_2^{\overline{\omega}}v_2w_2^{\omega}$. Then $u_1v_1w_1$ either occurs in $u_2^{\overline{\omega}}$ or w_2^{ω}. Hence, $u_1 = v_1 = w_1$ and similarly $u_2 = v_2 = w_2$. But $u_1^{\overline{\omega}}u_1^{\omega} = u_2^{\overline{\omega}}u_2^{\omega}$ implies that u_2^3 is a factor of u_1^4. Hence, the third condition from the lemma holds. □

Lemma 38. *Given two succinct expressions* $\mathbb{A}_1, \mathbb{A}_2$ *over* Σ *such that* $\mathsf{val}(\mathbb{A}_1)$ *and* $\mathsf{val}(\mathbb{A}_2)$ *are primitive words, we can decide in polynomial time whether* $\mathsf{val}(\mathbb{A}_1) = \mathsf{val}(\mathbb{A}_2)$.

Proof. We have to distinguish the following cases:

Case 1. $\mathsf{val}(\mathbb{A}_i)$ $(i \in \{1,2\})$ is finite. Then $\mathsf{val}(\mathbb{A}_1) = \mathsf{val}(\mathbb{A}_2)$ can be checked in polynomial time by Remark 33(d).

Case 2. $\mathsf{val}(\mathbb{A}_i)$ is Γ_i-uniform $(i \in \{1,2\})$. Then $\mathsf{val}(\mathbb{A}_1) = \mathsf{val}(\mathbb{A}_2)$ if and only if $\Gamma_1 = \Gamma_2$ which can be checked in polynomial time.

Case 3. $\mathsf{val}(\mathbb{A}_i) = u_iv_i^{\omega}$ $(i \in \{1,2\})$. By Lemma 36 we can produce SLPs for u_i and v_i $(i \in \{1,2\})$ from $\mathbb{A}_1$ and $\mathbb{A}_2$, respectively, in polynomial time. Let $k_i = |u_i|$ and $\ell_i = |v_i|$. Let $\mathsf{lcm}(\ell_1, \ell_2)$ denote the least common multiple of ℓ_1 and ℓ_2. By replacing v_i by $v_i^{\mathsf{max}(k_1,k_2)\cdot\mathsf{lcm}(\ell_1,\ell_2)/\ell_i}$ (for which we can compute an SLP in polynomial time by Remark 33(b)), we can assume that $|v_1| = |v_2| \geq \max\{k_1, k_2\}$. Let $\ell = |v_1| = |v_2|$. W.l.o.g assume that $k_1 \leq k_2$ and let $k = k_2 - k_1 \leq \ell$. If $k_1 < k_2$, then we can replace u_1 and v_1 by $u_1v_1[1:k]$ and $v_1[k+1:\ell]v_1[1:k]$, respectively (we can compute SLPs for these words in polynomial time by Remark 33(c)). Hence, we can also assume that $|u_1| = |u_2|$. But then, $u_1 v_1^\omega = u_2 v_2^\omega$ if and only if $u_1 = u_2$ and $v_1 = v_2$, which can be checked in polynomial time by Remark 33(d).

Case 4. $\mathsf{val}(\mathbb{A}_i) = u_i^{\overline{\omega}} v_i$ ($i \in \{1,2\}$). This case can be dealt with analogously to Case 3.

Case 5. $\mathsf{val}(\mathbb{A}_i) = u_i^{\overline{\omega}} v_i w_i^\omega$ ($i \in \{1,2\}$). By Lemma 36 we can produce SLPs for u_i, v_i, and w_i in polynomial time. As in Case 3, by replacing the words u_i, w_i by appropriate powers, we can enforce the condition $|u_1| = |u_2| = |w_1| = |w_2| = \ell \geq \max\{|v_1|, |v_2|\}$. In addition, we can enforce the condition $|v_1| = |v_2| = \ell$ as follows: If $k_i = |v_i| \leq \ell$, then we can replace v_i and w_i by $v_i w_i[1 : \ell - k_i]$ and $w_i[\ell - k_i + 1 : \ell] w_i[1 : \ell - k_i]$, respectively. Now, that we have $|u_1| = |u_2| = |v_1| = |v_2| = |w_1| = |w_2|$, we can check $u_1^{\overline{\omega}} v_1 w_1^\omega = u_2^{\overline{\omega}} v_2 w_2^\omega$ in polynomial time using Lemma 37 and Remark 33(e). □

4.6. A polynomial time equivalence test for succinct expressions

In this section, we will finally prove Theorem 35. The general strategy is very similar to [3]. We will incrementally reduce the $\omega\eta$-depth of the two given succinct expressions, until one of them (or both) describe primitive words. This allows to use the results from the previous section. We have to analyze carefully the size of the intermediate succinct expressions. In the following, Σ will always refer to a *finite* alphabet. A slight extension of SESs will be useful for the further consideration:

A *2-level system* is a tuple $\mathbb{A} = (\mathsf{Up}, \mathsf{Lo}, \Sigma, \mathsf{rhs})$ such that the following holds:

- The tuple $(\mathsf{Up}, \mathsf{Lo}, \mathsf{rhs}\restriction_{\mathsf{Up}})$ is an SES (w.l.o.g. in normal form) over the terminal alphabet Lo.
- The tuple $(\mathsf{Lo}, \Sigma, \mathsf{rhs}\restriction_{\mathsf{Lo}})$ is an SES over the terminal alphabet Σ.

The set Up (resp. Lo) is called the set of *upper level variables* (*lower level variables*) of $\mathbb{A}$. Moreover, we set $V = \mathsf{Up} \cup \mathsf{Lo}$ and call it the set of variables of $\mathbb{A}$. The SES $(\mathsf{Up}, \mathsf{Lo}, \mathsf{rhs}\restriction_{\mathsf{Up}})$ is called the *upper part of* $\mathbb{A}$, briefly $\mathsf{up}(\mathbb{A})$, and the SES $(\mathsf{Lo}, \Sigma, \mathsf{rhs}\restriction_{\mathsf{Lo}})$ is the *lower part of* $\mathbb{A}$, briefly, $\mathsf{lo}(\mathbb{A})$. The upper level evaluation mapping $\mathsf{uval}_{\mathbb{A}} : \mathsf{Up} \to \mathsf{Reg}(\mathsf{Lo})$ of $\mathbb{A}$ is defined as $\mathsf{uval}_{\mathbb{A}} = \mathsf{val}_{\mathsf{up}(\mathbb{A})}$. The evaluation mapping $\mathsf{val}_{\mathbb{A}}$ is defined by $\mathsf{val}_{\mathbb{A}}(X) = \mathsf{val}_{\mathsf{lo}(\mathbb{A})}(\mathsf{val}_{\mathsf{up}(\mathbb{A})}(X))$ for $X \in \mathsf{Up}$ and $\mathsf{val}_{\mathbb{A}}(X) = \mathsf{val}_{\mathsf{lo}(\mathbb{A})}(X)$ for $X \in \mathsf{Lo}$. Note that $\mathsf{val}_{\mathbb{A}}(X) = \mathsf{val}_{(\mathsf{Up} \cup \mathsf{Lo}, \Sigma, \mathsf{rhs})}(X)$ for all $X \in \mathsf{Up} \cup \mathsf{Lo}$. We will need certain nice properties of SESs and 2-level systems.

Definition 39 (primitive). *A primitive SES is an SES* $\mathbb{A} = (V, \Sigma, \mathsf{rhs})$ *such that* $\mathsf{val}_{\mathbb{A}}(X)$ *is primitive for all* $X \in V$*. A 2-level system* $\mathbb{B}$ *is primitive if* $\mathsf{lo}(\mathbb{B})$ *is primitive and for every upper level variable* X *of* $\mathbb{B}$*,* $\mathsf{val}(X)$ *is not primitive.*

Definition 40 (irredundant). *An irredundant SES is an SES* $\mathbb{A} = (V, \Sigma, \mathsf{rhs})$ *such that* $\mathsf{val}_{\mathbb{A}}(X) \neq \mathsf{val}_{\mathbb{A}}(Y)$ *for all* $X, Y \in V$ *with* $X \neq Y$*. A 2-level system* $\mathbb{B}$ *is irredundant if* $\mathsf{lo}(\mathbb{B})$ *is irredundant.*

One can think of a primitive and irredundant SES as a succinct representation of a finite subset of $D(\Sigma)$ where $\mathsf{val}_{\mathbb{A}} : V \to D(\Sigma)$ defines an injective mapping from V to this finite subset. Hence, for a regular word u such that all blocks belong to the image of $\mathsf{val}_{\mathbb{A}}$, we can define the $\mathsf{val}_{\mathbb{A}}$-skeleton of u. In the following, we will simply call it the $\mathbb{A}$-skeleton of u. A primitive and irredundant 2-level system intuitively is a system, where the terminal alphabet is a finite subset of $D(\Sigma)$ (namely the valuations of the variables of the lower part $\mathsf{lo}(\mathbb{B})$).

Remark 41. *If a primitive 2-level system* $\mathbb{B}$ *is not irredundant then, using Lemma 38, one can produce in polynomial time an irredundant 2-level system* $\mathbb{C}$ *such that* $\mathsf{val}(\mathbb{B}) = \mathsf{val}(\mathbb{C})$*. Indeed, if there are two different variables* $X, Y \in \mathsf{Lo}$ *such that* $\mathsf{val}_{\mathbb{B}}(X) = \mathsf{val}_{\mathbb{A}}(Y)$*, then one has to replace* X *in all right-hand sides by* Y*. Thereafter* X *can be removed from* Lo*. Note that this process does not change the set of upper level variables of* $\mathbb{B}$*.*

Assume that $\mathbb{B}$ is an SES or 2-level system and let $u = (A_i)_{i \in I}$ be a (possibly infinite) sequence of variables of $\mathbb{B}$. We say that u does not merge (is good, semi-good, irreducible with respect to R), if the sequence $(\mathsf{val}(A_i))_{i \in I}$ does not merge (is good, semi-good, irreducible with respect to R). Moreover, two sequences $u = (A_i)_{i \in I}$ and $v = (B_j)_{j \in J}$ of variables (possibly from two different SESs or 2-level systems) are equivalent if the sequences $(\mathsf{val}(A_i))_{i \in I}$ and $(\mathsf{val}(B_j))_{j \in J}$ are equivalent (i.e., $\prod_{i \in I} \mathsf{val}(A_i)$ and $\prod_{j \in J} \mathsf{val}(B_j)$ are isomorphic generalized words). The following definition is an adaption of the definition of a proper expression in [3].

Definition 42 (proper). *Let* $\mathbb{B} = (\mathsf{Up}, \mathsf{Lo}, \Sigma, \mathsf{rhs})$ *be a primitive 2-level system. A variable* $X \in \mathsf{Lo} \cup \mathsf{Up}$ *is proper if one of the following cases holds:*

(1) $X \in \mathsf{Lo}$

(2) $\mathsf{rhs}(X) = Y_1 \cdots Y_n$*, where* $Y_1, \dots, Y_n$ *are proper and* $Y_1 \cdots Y_n$ *does not merge.*

(3) $\mathsf{rhs}(X) = Y^\omega$ *or* $\mathsf{rhs}(X) = Y^{\overline{\omega}}$*, where* Y *is proper and* YYY *does not merge.*

(4) $\mathsf{rhs}(X) = [Y_1, \dots, Y_n]^\eta$ *where* $Y_1, \dots, Y_n$ *are proper and* $\mathsf{val}(X)$ *is not primitive.*

The 2-level system $\mathbb{B}$ *is proper if* $\mathbb{B}$ *is irredundant, primitive, and all variables are proper.*

Note that the condition that YYY does not merge in Definition 42(3) implies that $YYY\cdots$ and $\cdots YYY$ both do not merge by Lemma 23. Moreover, condition (4) from Definition 42 means that $Y_1, \dots, Y_n$ are proper and at least on $\mathsf{val}(Y_i)$ is not a single symbol.

The following Lemma follows directly from [3, Corollary 75]. Intuitively it says the following: Let $\mathbb{B}$ be a proper 2-level system and X an upper level variable. Then every block of the regular word $\mathsf{val}(X)$ is isomorphic to a unique primitive word $\mathsf{val}(Y)$ for some lower level nonterminal Y (uniqueness holds since $\mathbb{B}$ is irredundant. Moreover, if we replace every block B of $\mathsf{val}(X)$ by the unique lower level nonterminal Y such that B is isomorphic to $\mathsf{val}(Y)$, then we obtain the regular word $\mathsf{uval}(X)$.

Lemma 43. *Let* $\mathbb{B}$ *be a proper 2-level system and* X *an upper level variable. Then* $\mathsf{uval}(X)$ *is the* $\mathsf{lo}(\mathbb{B})$*-skeleton of* $\mathsf{val}(X)$*.*

The next two lemmas will be used to make a given 2-level system proper.

Lemma 44. *Given a primitive 2-level system* $\mathbb{B}$ *and a finite semi-good sequence* $A_1 \cdots A_m$ *of variables of* $\mathbb{B}$*, we can produce in polynomial time a primitive 2-level system* $\mathbb{C}$ *and a sequence* $B_1 \cdots B_n$ *of variables of* $\mathbb{C}$ *such that the following holds:*

- *The upper parts of* $\mathbb{B}$ *and* $\mathbb{C}$ *are the same, and the lower part of* $\mathbb{C}$ *extends the lower part of* $\mathbb{B}$ *by at most* $m-1$ *many new lower level variables, whose right-hand sides have length 2.*
- *The sequence* $B_1 \cdots B_n$ *is good.*

- *$A_1 \cdots A_m$ and $B_1 \cdots B_n$ are equivalent sequences.*

- *The subsequence of upper level variables in $A_1 \cdots A_m$ is the same as the subsequence of upper level variables in $B_1 \cdots B_n$.*

- *$n \leq m$.*

Proof. As long as the sequence $A_1 \cdots A_m$ contains a factor $A_i A_{i+1}$ or $A_i A_{i+1} A_{i+2}$, whose evaluation is a left-hand side of our rewriting system R, we do the following:

If $\mathsf{val}(A_i)$ is right-closed and $\mathsf{val}(A_{i+1})$ is left-closed, then we introduce a new lower level variable A, set $\mathsf{rhs}(A) = A_i A_{i+1}$, and replace the sequence $A_1 \cdots A_m$ by the sequence $A_1 \cdots A_{i-1} A A_{i+2} \cdots A_m$. If $\mathsf{val}(A_i) = \mathsf{val}(A_{i+1}) = \Gamma^\eta$ for some $\Gamma \subseteq \Sigma$, we continue with the sequence $A_1 \cdots A_{i-1} A_{i+1} \cdots A_m$. Finally, if $\mathsf{val}(A_i) = \mathsf{val}(A_{i+2}) = \Gamma^\eta$ for some $\Gamma \subseteq \Sigma$ and $\mathsf{val}(A_{i+1}) = a \in \Gamma$, we continue with the sequence $A_1 \cdots A_{i-1} A_{i+2} \cdots A_m$. We iterate this process as long as possible.

The resulting sequence $B_1 \cdots B_n$ is irreducible with respect to R and it is semi-good by Lemma 31. Hence it is good by Definition 29. □

Lemma 45. *Given a primitive SES $\mathbb{B}$ and a finite irreducible sequence $A_1 \cdots A_k$ ($k \geq 3$) of variables, we can produce in polynomial time a primitive SES $\mathbb{C}$ and sequences $B_1 \cdots B_m$, $C_1 \cdots C_n$ ($0 \leq m \leq k$, $1 \leq n \leq k$) of variables such that the following holds:*

- *$\mathbb{C}$ extends $\mathbb{B}$ by at most one new variable, whose right-hand side has length 2.*

- *The infinite sequence $B_1 \cdots B_m (C_1 \ldots C_n)^\omega$ is irreducible.*

- *$(A_1 \cdots A_k)^\omega$ and $B_1 \cdots B_m (C_1 \cdots C_n)^\omega$ are equivalent sequences.*

Proof. W.l.o.g. assume that $(A_1 \cdots A_k)^\omega$ is not irreducible. Since $A_1 \cdots A_k$ is irreducible, an R-reduction in the infinite sequence

$$A_1 \cdots A_k A_1 \cdots A_k A_1 \cdots A_k \cdots$$

can only occur at a border between A_k and A_1. There are the following cases, according to the left-hand sides of the system R.

Case 1. $\mathsf{val}(A_k) = \mathsf{val}(A_1) = \Gamma^\eta$ for some $\Gamma \subseteq \Sigma$. Then, the infinite sequence $A_1 A_2 \cdots A_k (A_2 \cdots A_k)^\omega$ is irreducible and equivalent to our original sequence (recall that $k \geq 3$).

Case 2. $\mathsf{val}(A_k)$ is scattered and right-closed, $\mathsf{val}(A_1)$ is scattered and left-closed. Then, we introduce a new lower level variable A with $\mathsf{rhs}(A) = A_k A_1$. It follows that the infinite sequence

$$A_1 A_2 \cdots A_{k-1} (A A_2 \cdots A_{k-1})^\omega$$

is irreducible and equivalent to our original sequence.

Case 3. $\mathsf{val}(A_k) = \Gamma^\eta$, $\mathsf{val}(A_1) = a$, $\mathsf{val}(A_2) = \Gamma^\eta$ for some $\Gamma \subseteq \Sigma$ and $a \in \Gamma$. If $k = 3$, then $A_1 A_2 \cdots A_k = A_1 A_2 A_3$ would not be irreducible (since $\mathsf{val}(A_2) = \mathsf{val}(A_3) = \Gamma^\eta$), which contradicts our assumptions. Hence, assume that $k \geq 4$. Then, the sequence $A_1 A_2 \cdots A_k (A_3 \cdots A_k)^\omega$ is again irreducible and equivalent to our original sequence.

Case 4. $\mathsf{val}(A_{k-1}) = \Gamma^\eta$, $\mathsf{val}(A_k) = a$, $\mathsf{val}(A_1) = \Gamma^\eta$ for some $\Gamma \subseteq \Sigma$ and $a \in \Gamma$. This case is similar to Case 3. □

Definition 46 (2-unfolded). *Let $\mathbb{B}$ be a primitive 2-level system. A variable $X \in \mathsf{Up}$ is called* 2-unfolded *if the following holds:*

(a) If $\mathsf{val}(X)$ has a first block, then $\mathsf{rhs}(X) = Au$ for some $A \in \mathsf{Lo}$ and expression u, and $\mathsf{val}(A)$ is the first block of $\mathsf{val}(X)$.

(b) If $\mathsf{val}(X)$ has a second block and the first block is scattered, then $\mathsf{rhs}(X) = ABu$ for some $A, B \in \mathsf{Lo}$ and expression u, and $\mathsf{val}(B)$ is the second block of $\mathsf{val}(X)$.

(c) If $\mathsf{val}(X)$ has a last block, then $\mathsf{rhs}(X) = uA$ for some $A \in \mathsf{Lo}$ and expression u, and $\mathsf{val}(A)$ is the last block of $\mathsf{val}(X)$.

(d) If $\mathsf{val}(X)$ has a second last block and the last block is scattered, then $\mathsf{rhs}(X) = uBA$ for some $A, B \in \mathsf{Lo}$ and expression u, and $\mathsf{val}(B)$ is the second last block of $\mathsf{val}(X)$.

We need the following two lemmas about 2-unfolded variables.

Lemma 47. *Let $\mathbb{B}$ be a primitive 2-level system and $X \in \mathsf{Up}$. If $\mathsf{rhs}(X) \in \mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$ and $\mathsf{rhs}(X)$ is good, then X is 2-unfolded.*

Proof. By symmetry let us only consider conditions (a) and (b) from Definition 46. Assume that $\mathsf{rhs}(X)$ is a good sequence. If $\mathsf{rhs}(X) \in \mathsf{Lo}^{\geq 2}$, then Lemma 30 implies that the variables in $\mathsf{rhs}(X)$ evaluate to the blocks of $\mathsf{val}(X)$ (recall that $\mathsf{rhs}(X)$ is good). Hence (a) and (b) hold. Next, assume that $\mathsf{rhs}(X) \in \mathsf{Lo}^{\geq 2}\mathsf{UpLo}^*$. Again, since $\mathsf{rhs}(X)$ is good, Lemma 30 implies that the first two variables in $\mathsf{rhs}(X)$ evaluate to the first two blocks of $\mathsf{val}(X)$. Thus, (a) and (b) hold again. If $\mathsf{rhs}(X) \in \mathsf{UpLo}^*$, then the first variable of $\mathsf{rhs}(X)$ evaluates to a non-primitive word. Since $\mathsf{rhs}(X)$ is good, it follows that $\mathsf{val}(X)$ does not have a first block (otherwise we get a contradiction to condition (2a) from Definition 29) and (a) and (b) hold. Finally assume that $\mathsf{rhs}(X) \in \mathsf{LoUpLo}^*$ and the first two variables of $\mathsf{rhs}(X)$ are $A \in \mathsf{Lo}$ and $Z \in \mathsf{Up}$. Then, $\mathsf{val}(A)$ is the first block of $\mathsf{val}(X)$, and (a) from Definition 46 holds. Since $\mathsf{rhs}(X)$ is good either $\mathsf{val}(Z)$ does not have a first block (and thus $\mathsf{val}(X)$ has no second block) or $\mathsf{val}(Z)$ has a first block but $\mathsf{val}(A)$ is uniform. In both cases (b) from Definition 46 is satisfied. This proves the lemma. □

Lemma 48. *Let $\mathbb{B}$ be a primitive 2-level system. Assume that $X_1, \ldots, X_n \in \mathsf{Up}$ are proper and 2-unfolded upper level variables such that either $\mathsf{rhs}(X_i) \in \mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{UpLo}^*$ or $\mathsf{rhs}(X_i)$ is a dense shuffle. Let $\bar{u}$ be a sequence of variables such that every upper level variable in $\bar{u}$ belongs to $\{X_1, \ldots, X_n\}$. Let the sequence $\bar{v}$ result from $\bar{u}$ by replacing every occurrence of a variable X_i with $\mathsf{rhs}(X_i) \in \mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{UpLo}^*$ by $\mathsf{rhs}(X_i)$. Then $\bar{v}$ is semi-good.*

Proof. Let $\bar{v} = (Z_j)_{j \in J}$. In order show that this sequence is semi-good, we consider an $j \in J$ such that $\mathsf{val}(Z_j)$ is not primitive (i.e., $Z_j \in \mathsf{Up}$) and has a first block (the condition from Definition 29 for the case that $\mathsf{val}(Z_j)$ has a last block can be verified analogously). Since $\mathsf{val}(Z_j)$ has a first block, Z_j cannot be one of the variables X_i ($1 \leq i \leq n$) such that $\mathsf{rhs}(X_i)$ is a dense shuffle (then $\mathsf{val}(Z_j)$ would be a non-primitive dense shuffle and would have no first block). Hence Z_j arises from replacing an occurrence of a variable X_i in $\bar{u}$ by $\mathsf{rhs}(X_i) \in \mathsf{Lo}^*\mathsf{UpLo}^*$. Let $\mathsf{rhs}(X_i) = A_1 \cdots A_m Z_j B_1 \cdots B_n$. Note that since X_i is proper by assumption, the sequence $A_1 \cdots A_m Z_j B_1 \cdots B_n$ does not merge.

Since $\mathsf{val}(Z_j)$ has a first block, also $\mathsf{val}(X_i)$ has a first block. Since X_i is 2-unfolded, we have $m \geq 1$. But then $\mathsf{val}(X_i)$ has also a second block. In order show condition (2a) from Definition 29, we have to show that ($m \geq 1$, $\mathsf{val}(A_m)$ is uniform, and $A_m Z_j$ does not merge) or ($m \geq 2$ and $A_{m-1} A_m Z_j$ does not merge). We already know that the whole sequence $A_1 \cdots A_m Z_j B_1 \cdots B_n$ does not merge. Hence, we only have to show that ($m \geq 1$ and $\mathsf{val}(A_m)$ is uniform) or $m \geq 2$.

Hence, if $m \geq 2$, we are done. So, assume that $m = 1$. Since $\mathsf{val}(X_i)$ has a second block, condition (b) from Definition 46 implies that $\mathsf{val}(A_1) = \mathsf{val}(A_m)$ is not scattered, i.e., it is uniform.

Let $\mathbb{B}$ be an SES (as usual in normal form) and X a variable with $\omega\eta\text{-}\mathsf{depth}(X) = h \geq 1$. Then there is a sequence of variables $X_1, \ldots, X_h$ such that the following properties hold (here, it is important that $\mathbb{B}$ is in normal form):

- $X_h \preceq_{\mathbb{B}} X$,
- $X_i \prec_{\mathbb{B}} X_{i+1}$ for $1 \leq i < h$,
- $\omega\eta\text{-}\mathsf{depth}(X_i) = i$ for $1 \leq i \leq h$, and
- $\mathsf{rhs}(X_i)$ is of the form Y^ω, $Y^{\overline{\omega}}$ or $[Y_1, \ldots, Y_n]^\eta$.

Note that $\mathsf{val}(X_1)$ is either primitive or a shuffle of finite words. If $\mathsf{val}(X_1) = [u_1, \ldots, u_k]^\eta$ where at least one of the u_i is in $\Sigma^{\geq 2}$ (thus, $\mathsf{val}(X_1)$ is not primitive), then the sequence $(X_1, \ldots, X_h)$ is called a *bad sequence.* If a variable X has a bad sequence, then we say it is of *bad shape.* Otherwise it is of *good shape.* For instance, if $\mathsf{rhs}(X) = [Y]^\eta$ and $\mathsf{rhs}(Y) = ab$, then X is of bad shape.

Proposition 49. *Let $\mathbb{B} = (V, \Sigma, \mathsf{rhs})$ be an SES such that for every variable $X \in V$, either $\mathsf{rhs}(X) \in \Sigma^+ \cup \Sigma^* V \Sigma^* \cup VV$ or $\mathsf{rhs}(X)$ is of the form Y^ω, $Y^{\overline{\omega}}$, or $[Y_1, \ldots, Y_n]^\eta$ for $Y, Y_1, \ldots, Y_n \in V \cup \Sigma$. Given $\mathbb{B}$ we can produce in polynomial time a proper 2-level system $\mathbb{C} = (\mathsf{Up}, \mathsf{Lo}, \Sigma, \mathsf{rhs})$ such that every variable $X \in V$, where $\mathsf{val}_{\mathbb{B}}(X)$ is not primitive, belongs to Up and for each of these variables X we have:*

(a) $\mathsf{val}_{\mathbb{B}}(X) = \mathsf{val}_{\mathbb{C}}(X)$

(b) If X is of good shape in $\mathbb{B}$, then $\omega\eta\text{-}\mathsf{depth}_{\mathbb{B}}(X) > \omega\eta\text{-}\mathsf{depth}_{\mathsf{up}(\mathbb{C})}(X)$.

(c) If X is of bad shape in $\mathbb{B}$, then $\omega\eta\text{-}\mathsf{depth}_{\mathbb{B}}(X) \geq \omega\eta\text{-}\mathsf{depth}_{\mathsf{up}(\mathbb{C})}(X)$ and X is of good shape in $\mathsf{up}(\mathbb{C})$.

Proof. W.l.o.g. we can assume that $\mathsf{val}(\mathbb{B})$ is not primitive. We start with some preprocessing.

Preprocessing.. First we transform the SES $\mathbb{B}$ into a primitve 2-level system $\mathbb{C}$ by collecting in Lo all variables X such that $\mathsf{val}(X)$ is primitive. This can be done in polynomial time using Lemma 36. Note that if $\mathsf{val}(X)$ is primitive and scattered, then for every Y in $\mathsf{rhs}(X)$, $\mathsf{val}(Y)$ is primitive too. But if $\mathsf{val}(X)$ is a primitive dense shuffle (i.e., of the form Γ^η for some $\Gamma \subseteq \Sigma$), then this is not necessarily true.[7] Hence, in this case we have to redefine $\mathsf{rhs}(Y) = \Gamma^\eta$. After this process the 2-level system $\mathbb{C}$ is already primitive and satisfies conditions (a), (b), and (c) from the proposition. All these properties will stay invariant throughout the remaining proof where we manipulate the system $\mathbb{C}$ in order to make it proper.

Before we come to the actual algorithm we transform $\mathbb{C}$ for technical convenience such that for all $X \in \mathsf{Up}$ one of the following holds:

(1) $\mathsf{rhs}(X) \in \mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$,

(2) $\mathsf{rhs}(X) = [Y_1, \dots Y_n]^\eta$ for some $Y_1, \dots, Y_n \in \mathsf{Up} \cup \mathsf{Lo}$,

(3) $\mathsf{rhs}(X) \in \mathsf{Up}^2$,

(4) $\mathsf{rhs}(X) = Y^\omega$ for $Y \in \mathsf{Up} \cup \mathsf{Lo}$,

(5) $\mathsf{rhs}(X) = Y^{\overline{\omega}}$ for $Y \in \mathsf{Up} \cup \mathsf{Lo}$.

In order to achieve this form we simply introduce for each upper level variable X with $\mathsf{rhs}(X) = uYv$ where $u, v \in \Sigma^*$ and $Y \in V$ two variables $X_u, X_v \in \mathsf{Lo}$ and set $\mathsf{rhs}(X) = X_u Y X_v$, $\mathsf{rhs}(X_u) = u$, and $\mathsf{rhs}(X_v) = v$ (if e.g. $u = \varepsilon$, then X_u is not present). Moreover, if a symbol $a \in \Sigma$ occurs in a right-hand side of the form Y^ω, $Y^{\overline{\omega}}$, or $[Y_1, \dots, Y_n]^\eta$, then we replace that occurrence by a new Lo-variable with right-hand side a. In fact, by this preprocessing all right-hand sides of the form (1) have length at most 3. This fact will be important when we estimate the size of the final system. From now on variables in Up that have a right-hand side of form (i) (for $1 \leq i \leq 5$) are said to be of type (i). Moreover, type (1, 2) means "type (1) or (2)" and type (3-5) means "type (3), (4) or (5)".

Following [3, proof of Theorem 65 and 66] we will now give an algorithm that produces a proper 2-level system. We will proceed along the hierarchical

[7] Let, for instance, $\mathsf{rhs}(X) = [Y]^\eta$ with $\mathsf{val}(Y) = a[a]^\eta$. Then $\mathsf{val}(X) = [a]^\eta$ is primitive but $\mathsf{val}(Y)$ is not primitive.

order of the variables in Up where in each step we possibly add new (lower und upper level) variables and change the right-hand sides of the original variables such that at the end, all upper level variables are proper and of the form (1)–(5). Moreover, all original upper variables are in addition 2-unfolded and of type (1, 2). Here "original" refers to the fact that the variable is already present in the initial 2-level system, in contrast to the new variables that are introduced during the process.

Actual algorithm.. We can now outline our procedure. Consider an original variable $X \in \mathsf{Up}$ such that every variable in $\mathsf{rhs}(X)$ is either in Lo or was already processed and is therefore now proper, 2-unfolded, and of type (1, 2). We need to distinguish on the form of the right-hand side of X. In all of the following cases, we reset $\mathsf{rhs}(X)$ (without changing $\mathsf{val}(X)$) either

(i) to a dense shuffle of variables that are already proper or

(ii) to a good sequence from $\mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$ of proper variables.

In (i), X is proper by Definition 42(4) (note that $\mathsf{val}(X)$ is not primitive since $X \in \mathsf{Up}$) and 2-unfolded since a non-primitive dense shuffle has no first or last block. In (ii) it follows from Lemma 30 and 47, that X is proper and 2-unfolded. For every new upper level variables Y that is introduced, the right-hand side is either

(i) a non-merging sequence of already proper variables or

(ii) Z^ω or $Z^{\overline{\omega}}$, where Z is already proper and ZZZ does not merge.

In both cases it follows from Definition 42 that Y is proper too.

In the following, we will several times make use of the following obvious fact: If a sequence $\bar{u} = (u_i)_{i\in I}$ of regular words does not merge and the sequence $\bar{v} = (v_i)_{i\in I}$ results from replacing some factors $(u_k)_{k\in K}$ (where K is an interval of I) by the single word $\prod_{k\in K} u_k$, then $\bar{v}$ does not merge as well.

Recall also that for every original upper level variable X with $\mathsf{rhs}(X) \in \mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$ we have $|\mathsf{rhs}(X)| \leq 3$ by our preprocessing.

Case 1. $\mathsf{rhs}(X) \in \mathsf{Lo}^2 \cup \mathsf{Lo}^3$ (hence $\mathsf{rhs}(X)$ is semi-good). By applying Lemma 44 to $\mathsf{rhs}(X)$, we can compute in polynomial time an equivalent good sequence of at most three possibly new Lo-variables (and their corresponding right-hand sides). This sequence becomes the new right-hand side of X.

Case 2. $\mathsf{rhs}(X) \in \mathsf{Lo}^{\leq 1}\mathsf{Up}\mathsf{Lo}^{\leq 1}$. Let Y be the unique Up-variable in $\mathsf{rhs}(X)$. Note that Y is one of the original variables, which has already been processed and hence is proper, 2-unfolded, and of type (1, 2). If $\mathsf{rhs}(Y) \in \mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$, then we replace Y in $\mathsf{rhs}(X)$ by $\mathsf{rhs}(Y)$. If $\mathsf{rhs}(Y)$ is a dense shuffle, then we leave Y in $\mathsf{rhs}(X)$. Since Y is proper and 2-unfolded, Lemma 48 implies that the resulting new right-hand side of X is semi-good and in $\mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$. Thus, we can apply Lemma 44 and obtain an equivalent good sequence in $\mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$ (as in Case 1, we will introduce new Lo-variables thereby). This good sequence will be the new right-hand side of X. For later estimating the length of right-hand sides, let us note the following points (where $\mathsf{rhs}(X)$ refers to the new right-hand side of X):

- $|\mathsf{rhs}(X)| \leq |\mathsf{rhs}(Y)| + 2$, and
- only new lower level variables are introduced.

Case 3. $\mathsf{rhs}(X) = [Y_1, \ldots, Y_k]^\eta$. Then there is nothing to do.

Case 4. $\mathsf{rhs}(X) = YZ$ for some $Y, Z \in \mathsf{Up}$. Here Y and Z are original variables, which have already been processed and therefore are proper, 2-unfolded, and of type (1, 2). If $\mathsf{rhs}(Y) \in \mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$ then we replace Y in YZ by $\mathsf{rhs}(Y)$. If $\mathsf{rhs}(Y)$ is a dense shuffle, we leave Y in YZ. We proceed analogously with Z in YZ. Since Y and Z are proper and 2-unfolded, Lemma 48 implies that the resulting new right-hand side of X is semi-good and contains at most two variables from Up. Thus we can apply Lemma 44 and obtain an equivalent good sequence u of variables with at most two variables from Up (again, we introduce new Lo-variables thereby).

Now, we replace parts in the sequence u in order to get $\mathsf{rhs}(X)$. First, assume that

$$u = A_1 \cdots A_k \in \mathsf{Lo}^{\geq 2}.$$

If $k \leq 5$, then $\mathsf{rhs}(X)$ simply becomes u (which is good). If $k \geq 6$, then we introduce a new Up-variable U and set

$$\mathsf{rhs}(X) = A_1 A_2 U A_{k-1} A_k, \quad \mathsf{rhs}(U) = A_3 \cdots A_{k-2}.$$

Since u is good, both right-hand sides are good as well. Second, assume that

$$u = A_1 \cdots A_k U B_1 \cdots B_\ell \in \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$$

with $U \in \mathsf{Up}$ proper. If $k \leq 2$ and $\ell \leq 2$ then we we simply set $\mathsf{rhs}(X) = u$. On the other hand, if $k > 2$ or $\ell > 2$, then we introduce a new Up-variable W of type (1) and set

$$\mathsf{rhs}(X) = A_1 A_2 W B_{\ell-1} B_\ell, \quad \mathsf{rhs}(W) = A_3 \cdots A_k U B_1 \cdots B_{\ell-2}$$

(if e.g. $k > 2$ but $\ell = 1$, then $B_1 \cdots B_{\ell-2}$ and $B_{\ell-1}$ disappear). Since u is good, $\mathsf{rhs}(X)$ will be good too. Moreover, since u does not merge (by Lemma 30), $\mathsf{rhs}(W)$ does not merge as well ($\mathsf{rhs}(W)$ is not necessarily good; but this is not required since W is a new variable). Third, assume that

$$u = A_1 \cdots A_k U B_1 \cdots B_\ell V C_1 \cdots C_n \in \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$$

with $U, V \in \mathsf{Up}$ proper. In this case we introduce two new Up-variables W (of type (1)) and W' (of type (3)) and set

$$\mathsf{rhs}(X) = A_1 A_2 W' C_1 \cdots C_n, \;\; \mathsf{rhs}(W') = WV, \;\; \mathsf{rhs}(W) = A_3 \cdots A_k U B_1 \ldots B_\ell.$$

Again, since u is good, $\mathsf{rhs}(X)$ is good as well. Moreover, since u does not merge, neither $\mathsf{rhs}(W)$ nor $\mathsf{rhs}(W')$ merges. Note that if $\mathsf{rhs}(Z)$ is a dense shuffle then $Z = V$ and the number n in the right-hand side of X is 0. On the other hand, if $\mathsf{rhs}(Z) \in \mathsf{Lo}^{\geq 2} \cup \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$ then n can be bounded by the length of the sequence $\mathsf{rhs}(Z)$. Hence, for Case 4 we

- turned the variable X of type (3) into a variable of type (1) with $|\mathsf{rhs}(X)| \leq \max\{5, 3 + |\mathsf{rhs}(Z)|\}$, and
- added at most one new upper level variable of type (3) and at most one new upper level variable W of type (1) with $|\mathsf{rhs}(W)| \leq |\mathsf{rhs}(Y)| + |\mathsf{rhs}(Z)|$.

Case 5. $\mathsf{rhs}(X) = Y^\omega$. Note that Y is either a Lo-variable, or it is an original Up-variable, which has already been processed and hence is proper, 2-unfolded, and of type (1, 2). We can therefore distinguish the following subcases.

Case 5(a). $\mathsf{rhs}(Y) = [Z_1, \ldots, Z_n]^\eta$ for some $Z_1, \ldots, Z_n \in \mathsf{Lo} \cup \mathsf{Up}$ proper. Then by the general identity $(\Gamma^\eta)^\omega \cong \Gamma^\eta$ (which follows from Cantor's theorem), we have $\mathsf{val}(X) = \mathsf{val}(Y)$. Hence $\mathsf{val}(Y)$ is a non-primitive dense shuffle. We set $\mathsf{rhs}(X) = [Z_1, \ldots, Z_n]^\eta$. The variable Y could be eliminated from the

2-level system (but this is not necessary). We turned a variable of type (4) into a variable of type (2).

Case 5(b). $\mathsf{rhs}(Y) \in \mathsf{Lo}^*\mathsf{Up}\mathsf{Lo}^*$. Let $\mathsf{rhs}(Y) = uZv$ with $Z \in \mathsf{Up}$ proper and $u, v \in \mathsf{Lo}^*$. Since Y is proper and 2-unfolded, Lemma 48 implies that the infinite sequence $uZvuZv \cdots = u(Zvu)^\omega$ is semi-good. By applying Lemma 44 to the sequence vu of Lo-variables, we obtain an equivalent good sequence $u(Zw)^\omega$. Here w is a sequence of (possibly new) Lo-variables such that w represents the irreducible normal form with respect to R of the sequence represented by vu. Note that $|w| \leq |uv|$. We set

$$\mathsf{rhs}(X) = uV, \quad \mathsf{rhs}(V) = U^\omega, \quad \mathsf{rhs}(U) = Zw.$$

Since the sequence $u(Zw)^\omega$ is good, also the sequence uV is good. Moreover, since $u(Zw)^\omega$ does not merge (by Lemma 30), the same holds for $\mathsf{rhs}(U) = Zw$ and UUU (so U and V are proper by definition). Let us note the following points:

- We turned the variable X of type (4) into a variable of type (1) with $|\mathsf{rhs}(X)| \leq |\mathsf{rhs}(Y)|$.
- We introduced a fresh variable V of type (4) and a fresh variable U of type (1) with $|\mathsf{rhs}(U)| \leq |\mathsf{rhs}(Y)|$.

Case 5(c). $Y \in \mathsf{Lo}$ and hence $\mathsf{val}(Y)$ is primitive. Then the infinite sequence $YYY \cdots$ must be irreducible, because otherwise $\mathsf{val}(Y)$ would be either finite or uniform and $\mathsf{val}(X) = \mathsf{val}(Y)^\omega$ would be primitive. We introduce a new Up-variable Z and set

$$\mathsf{rhs}(X) = YYZ, \quad \mathsf{rhs}(Z) = Y^\omega.$$

Then $\mathsf{rhs}(X)$ is good and YYY does not merge. Note that

- we turned the variable X of type (4) into a variable of type (1) with $|\mathsf{rhs}(X)| = 3$, and
- added a fresh variable of type (4).

Case 5(d). $\mathsf{rhs}(Y) \in \mathsf{Lo}^2$. Let $\mathsf{rhs}(Y) = A_1A_2$ for $A_1, A_2 \in \mathsf{Lo}$. Since Y is already proper, we know that A_1A_2 is irreducible. If the infinite sequence

$A_1A_2A_1A_2\cdots$ is irreducible too, then we introduce a fresh Up-variables Z and set

$$\mathsf{rhs}(X) = A_1A_2Z, \quad \mathsf{rhs}(Z) = Y^\omega.$$

Clearly, the sequence $\mathsf{rhs}(X)$ is good and YYY does not merge. As in Case 5(c), we

- turned the variable X of type (4) into a variable of type (1) with $|\mathsf{rhs}(X)| = 3$, and
- added a fresh variable of type (4).

On the other hand, if $A_1A_2A_1A_2\cdots$ is not irreducible, then (since A_1A_2 is irreducible), an R-reduction can only occur at a border between A_2 and A_1. The case that $\mathsf{val}(A_1) = \mathsf{val}(A_2) = \Gamma^\eta$ for some $\Gamma \subseteq \Sigma$ cannot occur (since A_1A_2 is irreducible). If $\mathsf{val}(A_2)$ is scattered and right-closed and $\mathsf{val}(A_1)$ is scattered and left-closed, then we introduce a fresh Lo-variable B and a fresh Up-variable Z and set

$$\mathsf{rhs}(X) = A_1BZ, \quad \mathsf{rhs}(Z) = B^\omega, \quad \mathsf{rhs}(B) = A_2A_1.$$

It is straightforward to show that the infinite sequence $A_1BBB\cdots$ is irreducible. Hence $\mathsf{rhs}(X)$ is good and BBB does not merge. Again, we

- turned the variable X of type (4) into a variable of type (1) with $|\mathsf{rhs}(X)| = 3$, and
- added a fresh variable of type (4).

Next, if $\mathsf{val}(A_1) = \Gamma^\eta$ and $\mathsf{val}(A_2) = a$ for some $\Gamma \subseteq \Sigma$ and $a \in \Gamma$, then $A_1A_2A_1A_2\cdots$ evaluates to Γ^η. Hence, $\mathsf{val}(X)$ is primitive, which is a contradiction. Finally, if $\mathsf{val}(A_2) = \Gamma^\eta$ and $\mathsf{val}(A_1) = a \in \Gamma$, then $A_1A_2A_1A_2\cdots$ evaluates to $a\Gamma^\eta = \mathsf{val}(Y)$ and we set

$$\mathsf{rhs}(X) = A_1A_2, .$$

which is good.

Case 5(e). $\mathsf{rhs}(Y) \in \mathsf{Lo}^{\geq 3}$. We apply Lemma 45 to the irreducible sequence $\mathsf{rhs}(Y)$ and compute finite sequences u, v of (possibly new) Lo-variables with their corresponding right-hand sides. We have $|u| \leq |\mathsf{rhs}(Y)|$ and $|v| \leq |\mathsf{rhs}(Y)|$. Moreover, the infinite sequence uv^ω of Lo-variables is irreducible (and hence good) and evaluates to $\mathsf{val}(X)$. W.l.o.g. we can assume $|u| \geq 2$ (otherwise, we can replace u by uvv). We introduce fresh Up-variables U and V and set

$$\mathsf{rhs}(X) = uV, \quad \mathsf{rhs}(V) = U^{\omega}, \quad \mathsf{rhs}(U) = v.$$

If $|v| = 1$, i.e., v consists of a single Lo-variable, then we do not need U. Then $\mathsf{rhs}(X)$ is good and UUU does not merge. We

- turned the variable X of type (4) into a variable of type (1) with $|\mathsf{rhs}(X)| \leq |\mathsf{rhs}(Y)| + 1$, and
- added a fresh variable V of type (4) and at most one fresh variable U of type (1).

Case 6. $\mathsf{rhs}(X) = Y^{\overline{\omega}}$. This case is symmetric to Case 4.

The resulting system $\mathbb{C}$ is primitive and all Up-variables are proper. On the other hand, $\mathbb{C}$ is not necessarily irredundant. But this can be easily achieved as described in Remark 41. □

We are now in the position to prove Theorem 35.

Proof of Theorem 35. It suffices to show that the following problem can be solved in polynomial time:

INPUT: An SES $\mathbb{A}$ and two variables X, Y of $\mathbb{A}$.
QUESTION: $\mathsf{val}(X) \cong \mathsf{val}(Y)$?

If both variables X and Y evaluate to primitive words, then we just need to apply Lemma 38. If only one of the two evaluates to a primitive word, then $\mathsf{val}(X) \ncong \mathsf{val}(Y)$. Hence, we may assume that both $\mathsf{val}(X)$ and $\mathsf{val}(Y)$ are not primitive. In particular, we have $\omega\eta\text{-}\mathsf{depth}(X) > 0$ and $\omega\eta\text{-}\mathsf{depth}(Y) > 0$. It is easy to bring $\mathbb{A}$ into the normal form required in Proposition 49. Applying Proposition 49 to $\mathbb{A}$ gives a proper 2-level system $\mathbb{A}_0$. The variables X and Y belong to the upper level part of $\mathbb{A}_0$. Starting with $\mathbb{A}_0$ we construct a sequence of proper 2-level systems $\mathbb{A}_j = (\mathsf{Up}_j, \mathsf{Lo}_j, \mathsf{Lo}_{j-1}, \mathsf{rhs}_j)$ (with $\mathsf{Lo}_{-1} = \Sigma$). In order to obtain $\mathbb{A}_j$ we apply the procedure of Proposition 49 to the SES $\mathsf{up}(\mathbb{A}_{j-1})$. Let k be maximal such that X and Y belong to the upper level part of $\mathbb{A}_k$. Since by Proposition 49 in every second step the $\omega\eta$-depth of X and Y strictly decreases we have $k \leq 2 \cdot |\mathbb{A}|$.

Let $0 \leq j \leq k$. By Lemma 43 $\mathsf{uval}_j(X)$ is the $\mathsf{lo}(\mathbb{A}_j)$-skeleton of $\mathsf{val}_j(X)$ and similarly for Y. Hence $\mathsf{val}_j(X) \cong \mathsf{val}_j(Y)$ if and only if $\mathsf{uval}_j(X) \cong$

$\mathsf{uval}_j(Y)$ by Proposition 20. Recall that $\mathbb{A}_{j+1}$ is obtained by applying the procedure of Proposition 49 to $\mathsf{up}(\mathbb{A}_j)$. We obtain $\mathsf{val}_j(X) \cong \mathsf{val}_j(Y)$ if and only if $\mathsf{val}_{j+1}(X) \cong \mathsf{val}_{j+1}(Y)$ for all $0 \leq j < k$. Hence, $\mathsf{val}(X) \cong \mathsf{val}(Y)$ if and only if $\mathsf{val}_k(X) \cong \mathsf{val}_k(Y)$ if and only if $\mathsf{uval}_k(X) \cong \mathsf{uval}_k(Y)$. Now, by the maximality of k, $\mathsf{uval}_k(X)$ or $\mathsf{uval}_k(Y)$ must be primitive. Hence, using Lemma 38, we can check in polynomial time whether $\mathsf{uval}_k(X) \cong \mathsf{uval}_k(Y)$.

Runtime.. Let us analyze the system $\mathsf{up}(\mathbb{A}_j)$ for $1 \leq j \leq k$. The 2-level system $\mathbb{A}_j$ is obtained by applying Proposition 49 to $\mathsf{up}(\mathbb{A}_{j-1})$. Since all right-hand sides of $\mathsf{up}(\mathbb{A}_{j-1})$ are of the form (1–5), the right-hand sides of $\mathsf{up}(\mathbb{A}_{j-1})$ have the form that we require in Proposition 49. Let $\mathsf{Type(3\text{-}5)}_j$ be the set of variables in Up_j that are of type (3-5).

Now let us estimate the number $|\mathsf{Up}_j|$ for $1 \leq j \leq k$. Observe that in the proof of Proposition 49 in each of the Cases 1–3 only new lower level variables are introduced. In each of the Cases 4–6 an original variable of type (3-5) is turned into a variable of type (1, 2) and at most one fresh variable of type (3-5) is added to Up_j. Moreover, additionally at most one fresh variables of type (1, 2) is added to Up_j. We conclude that $|\mathsf{Type(3\text{-}5)}_j| \leq |\mathsf{Type(3\text{-}5)}_{j-1}|$ and the total number of variables in Up_j is bounded by $|\mathsf{Up}_{j-1}| + 2 \cdot |\mathsf{Type(3\text{-}5)}_{j-1}|$. Recall that $j \leq k \leq 2|\mathbb{A}|$. Hence, for all $0 \leq j \leq k$ we get

$$|\mathsf{Up}_j| \leq |\mathsf{Up}_0| + 2j \cdot |\mathsf{Type(3\text{-}5)}_0| \leq |\mathbb{A}_0| \cdot (4 \cdot |\mathbb{A}| + 1).$$

Let us now estimate the maximal size of right-hand sides of $\mathbb{A}_j$. If $\mathsf{rhs}_j(X)$ is of the form Y^ω or $Y^{\overline{\omega}}$ then by definition $|\mathsf{rhs}_j(X)| = 2$. If $\mathsf{rhs}_j(X) = [Y_1, \dots, Y_n]^\eta$, where every Y_i is a variable or terminal symbol of $\mathbb{A}_j$, then $|\mathsf{rhs}_j(X)|$ is bounded by one plus the number of different variables or terminal symbols of $\mathbb{A}_j$. We will see in a moment that this number is bounded polynomially in $|\mathbb{A}|$.

Let us now consider the case that $\mathsf{rhs}_j(X)$ is a sequence of variables and terminal symbols. First, consider the case that $X \in \mathsf{Up}_j \cap \mathsf{Up}_{j-1}$, i.e., X is an upper level variable of $\mathbb{A}_j$ that was already an upper level variable of the previous system $\mathbb{A}_{j-1}$ (these variables are called the original variables in the proof of Proposition 49). In each of the six cases in the proof of Proposition 49, we have $|\mathsf{rhs}_j(X)| \leq \max\{5, 3 + |\mathsf{rhs}_j(Y)|\}$, where $Y \in \mathsf{Up}_j \cap \mathsf{Up}_{j-1}$ is a variable that was processed before X. We therefore obtain $|\mathsf{rhs}_j(X)| \leq 3 \cdot |\mathsf{Up}_j \cap \mathsf{Up}_{j-1}| + 5$. Hence, $|\mathsf{rhs}_j(X)| \leq 3 \cdot |\mathbb{A}_0| \cdot (4 \cdot |\mathbb{A}| + 1) + 5$. For a fresh variable $X \in \mathsf{Up}_j \setminus \mathsf{Up}_{j-1}$, we can bound $|\mathsf{rhs}_j(X)|$ by $2 \cdot \max\{|\mathsf{rhs}_j(Y)| \mid Y \in \mathsf{Up}_j \cap \mathsf{Up}_{j-1}\}$ (the factor 2 comes from Case 4). Hence $|\mathsf{rhs}_j(X)| \leq 6 \cdot |\mathbb{A}_0| \cdot (4 \cdot |\mathbb{A}| + 1) + 10$ for all $X \in \mathsf{Up}_j$. Finally, note that $|\mathbb{A}_0|$ is polynomially bounded in $|\mathbb{A}|$.

Next, consider a lower level variable A of $\mathbb{A}_j$ such that $\mathsf{rhs}_j(A)$ is a sequence of variables and terminal symbols. Then, $|\mathsf{rhs}_j(A)|$ is bounded by 2 (if A is introduced in one of the Cases 1–6 in the proof of Proposition 49) or by the maximal length of a right-hand side of a variable from $\mathbb{A}_{j-1}$ (if A is introduced in the preprocessing step).

Finally, notice that in each of the Cases 1–6, the number of fresh lower level variables that are introduced is bounded by $2 \cdot \max\{|\mathsf{rhs}_j(Y)| \mid Y \in \mathsf{Up}_j \cap \mathsf{Up}_{j-1}\}$ (the factor 2 comes again from Case 4). Hence the number of lower level variables is also bounded polynomially in $|\mathbb{A}|$.

We have shown that the total size of very 2-level system $\mathbb{A}_j$ $(1 \leq j \leq k)$ is bounded polynomially in $|\mathbb{A}|$. As the time needed to construct $\mathbb{A}_{j+1}$ from $\mathbb{A}_j$ is polynomially bounded by Proposition 49, we conclude that the overall running time of our algorithm is polynomially bounded as well. □

4.7. Deciding the existence of a non-trivial automorphism

A *non-trivial automorphism* of a generalized word $(L; \leq, \tau)$ is an automorphism of $(L; \leq, \tau)$, i.e., an isomorphism $f : (L; \leq, \tau) \to (L; \leq, \tau))$ such that $f(p) \neq p$ for at least one $p \in L$.

Remark 50. *Note that every automorphism f of a regular word w has to map blocks of w to blocks of w. Hence, f induces an automorphism of the skeleton of w. Thus, w has a non-trivial automorphism if and only if (i) there is a block of w having a non-trivial automorphism, or (ii) the skeleton of w has a non-trivial automorphism.*

Kuske has shown in [21] that it is decidable whether a given regular word has a non-trivial automorphism. Using our machinery for the isomorphism problem we can easily show the following result:

Theorem 51. *For a given partitioned DFA $\mathcal{A}$ it can be decided in polynomial time whether $w(\mathcal{A})$ has a non-trivial automorphism.*

Before we prove Theorem 51, let us first consider the case of a succinctly specified primitive word.

Lemma 52. *For a given succinct expression $\mathbb{A}$ such that* $\mathsf{val}(\mathbb{A})$ *is primitive, one can decided in polynomial time whether* $\mathsf{val}(\mathbb{A})$ *has a non-trivial automorphism.*

PROOF. By Lemma 36 we can determine in polynomial time, whether $\mathsf{val}(\mathbb{A})$ is finite, Γ-uniform (for some finite subalphabet Γ), or of the form $u^{\overline{\omega}}v$, vw^{ω}, or $u^{\overline{\omega}}vw^{\omega}$ ($u, w \in \Sigma^+$, $v \in \Sigma^*$). Clearly, a finite word, or a regular word of the form uv^{ω} or $u^{\overline{\omega}}v$ has no non-trivial automorphism. If $\mathsf{val}(\mathbb{A})$ is Γ-uniform, then $\mathsf{val}(\mathbb{A})$ has non-trivial automorphisms. Finally, assume that $\mathsf{val}(\mathbb{A})$ is of the form $u^{\overline{\omega}}vw^{\omega}$ for finite words u, v, and w. By Lemma 36 we can compute in polynomial time SLPs $\mathbb{B}$, $\mathbb{C}$, and $\mathbb{D}$ such that $u = \mathsf{val}(\mathbb{B})$, $v = \mathsf{val}(\mathbb{C})$, and $w = \mathsf{val}(\mathbb{D})$. The word $u^{\overline{\omega}}vw^{\omega}$ has a non-trivial automorphism if and only if $u^{\overline{\omega}}vw^{\omega} \cong u^{\overline{\omega}}u^{\omega}$. Hence, we have to check whether $\mathsf{val}(\mathbb{A}) \cong \mathsf{val}(\mathbb{B})^{\overline{\omega}}\mathsf{val}(\mathbb{B})^{\omega}$, which can be checked in polynomial time by Lemma 38. □

Proof of Theorem 51. By Theorem 34, it suffices to show that for a given SES $\mathbb{A}$ one can check in polynomial time whether $\mathsf{val}(\mathbb{A})$ has a non-trivial automorphism. From $\mathbb{A}$ we can compute by Proposition 49 in polynomial time a proper 2-level system $\mathbb{A}_0$ and a variable X of $\mathbb{A}_0$ such that $\mathsf{val}(\mathbb{A}) = \mathsf{val}_{\mathbb{A}_0}(X)$. By Lemma 52 we can assume that X is an upper level variable of $\mathbb{A}_0$. Define the proper 2-level systems $\mathbb{A}_j$ as in the proof of Theorem 35. Let $k > 0$ be minimal such that X is a lower level variable of $\mathbb{A}_k$. In the proof of Theorem 35 we have shown that the 2-level systems $\mathbb{A}_0, \ldots, \mathbb{A}_k$ can be all computed in polynomial time. By Lemma 43, $\mathsf{uval}_{\mathbb{A}_j}(X)$ is the $\mathsf{lo}(\mathbb{A}_j)$-skeleton of $\mathsf{val}_{\mathbb{A}_j}(X)$ for all $0 \leq j < k$.

Let $0 \leq j < k$. Let L_j be the set of all lower level variables Y of $\mathbb{A}_j$ such that (i) $Y \preceq_{\mathbb{A}_j} X$ and (ii) Y appears in a right-hand side of $\mathsf{up}(\mathbb{A}_j)$. Then, $\mathsf{uval}_{\mathbb{A}_j}(X)$ is still the L_j-skeleton of $\mathsf{val}_{\mathbb{A}_j}(X)$. Moreover, for every $Y \in L_j$, there is a block of $\mathsf{val}_{\mathbb{A}_j}(X)$ that is isomorphic to $\mathsf{val}_{\mathbb{A}_j}(Y)$. By Remark 50, $\mathsf{val}_{\mathbb{A}_j}(X)$ has a non-trivial automorphism if and only if (i) there is $Y \in L_j$ such that $\mathsf{val}_{\mathbb{A}_j}(Y)$ has a non-trivial automorphism or (ii) $\mathsf{uval}_{\mathbb{A}_j}(X) = \mathsf{val}_{\mathbb{A}_{j+1}}(X)$ has a non-trivial automorphism. By Lemma 52 we can check property (i) in polynomial time. Moreover, since $\mathsf{val}_{\mathbb{A}_k}(X)$ is primitive, we can also check (ii) for $j = k-1$ in polynomial time. This gives a polynomial time algorithm for verifying whether $\mathsf{val}_{\mathbb{A}_0}(X) = \mathsf{val}(\mathbb{A})$ has a non-trivial automorphism. □

4.8. Lower bounds for regular linear orders

In this section we prove lower bounds for the isomorphism problem for regular words. In fact, all these lower bounds only need a unary alphabet, i.e., they hold for regular linear orders. We identify in the following the linear order $(L; \leq)$ we the generalized word $(L; \leq, \tau)$ where $\tau(x) = a$ for all $x \in L$.

In particular, for $n \in \mathbb{N}$ the regular expression

$$\underbrace{a^\omega a^\omega \cdots a^\omega}_{n \text{ times}}.$$

is identified with the linear order $\omega \cdot \mathbf{n}$. Moreover, the regular expression $\alpha\beta$ denotes the order sum $\alpha + \beta$ of α and β (viewed as 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 53. *The following problem is* P*-hard (and hence* P*-complete) for every finite alphabet* Σ*:*

INPUT: Two succinct expressions $\mathbb{A}_1$ *and* $\mathbb{A}_2$ *over the alphabet* Σ.
QUESTION: $\mathsf{val}(\mathbb{A}_1) \cong \mathsf{val}(\mathbb{A}_2)$*?*

Proof. Note that the problem can be solved in polynomial time by Theorem 35. P-hardness will be shown by a reduction from the monotone circuit value problem. So, let C be a monotone boolean circuit. We can assume that the gates of C are partitioned into layers $L_1, \ldots, L_n$, where L_1 contains all input gates, L_n only contains the output gate, and all inputs for a gate from L_{i+1} belong to L_i. Moreover, L_i ($i > 1$) either contains only and-gates or or-gates. We construct an SES $\mathbb{A}$ (over a unary terminal alphabet $\{a\}$), which contains for each gate v of C a variable test_v and for each layer $d \in \{1, \ldots, n\}$ two variables good_d, and bad_d such that the following holds for all gates $v \in L_d$:

(a) Either $\mathsf{val}_{\mathbb{A}}(\mathsf{test}_v) \cong \mathsf{val}_{\mathbb{A}}(\mathsf{bad}_d)$ or $\mathsf{val}_{\mathbb{A}}(\mathsf{test}_v) \cong \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)$.

(b) $\mathsf{val}_{\mathbb{A}}(\mathsf{test}_v) \cong \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)$ if and only if gate v evaluates to true.

(c) The linear orders $\mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)$ and $\mathsf{val}_{\mathbb{A}}(\mathsf{bad}_d)$ are shuffles that do not contain an interval isomorphic to $\omega \cdot \mathbf{d}$.

The base case for the first layer is trivial. Set $\mathsf{rhs}_{\mathbb{A}}(\mathsf{good}_1) = a$ and $\mathsf{rhs}_{\mathbb{A}}(\mathsf{bad}_1) = aa$. In other words, $\mathsf{val}_{\mathbb{A}}(\mathsf{good}_1) \cong \mathbf{1}$ and $\mathsf{val}_{\mathbb{A}}(\mathsf{bad}_1) \cong \mathbf{2}$. Moreover, $\mathsf{rhs}_{\mathbb{A}}(\mathsf{test}_v) = a$ if $v \in L_1$ is a true-gate and $\mathsf{rhs}_{\mathbb{A}}(\mathsf{test}_v) = aa$ if $v \in L_1$ is a false-gate.

Now assume that $v \in L_{d+1}$ is a gate with inputs $v_1, v_2 \in L_d$. There are two cases:

Case 1. L_{d+1} consists of and-gates. Then we set

$$\begin{aligned}
\mathsf{rhs}_{\mathbb{A}}(\mathsf{test}_v) &= [\omega \cdot \mathbf{d} + \mathsf{test}_{v_1}, \omega \cdot \mathbf{d} + \mathsf{test}_{v_2}, \omega \cdot \mathbf{d} + \mathsf{good}_d]^\eta \\
\mathsf{rhs}_{\mathbb{A}}(\mathsf{good}_{d+1}) &= [\omega \cdot \mathbf{d} + \mathsf{good}_d]^\eta \\
\mathsf{rhs}_{\mathbb{A}}(\mathsf{bad}_{d+1}) &= [\omega \cdot \mathbf{d} + \mathsf{good}_d, \omega \cdot \mathbf{d} + \mathsf{bad}_d]^\eta.
\end{aligned}$$

Case 2. L_{d+1} consists of or-gates.

$$\begin{aligned}
\mathsf{rhs}_{\mathbb{A}}(\mathsf{test}_v) &= [\omega \cdot \mathbf{d} + \mathsf{test}_{v_1}, \omega \cdot \mathbf{d} + \mathsf{test}_{v_2}, \omega \cdot \mathbf{d} + \mathsf{bad}_d]^\eta \\
\mathsf{rhs}_{\mathbb{A}}(\mathsf{good}_{d+1}) &= [\omega \cdot \mathbf{d} + \mathsf{good}_d, \omega \cdot \mathbf{d} + \mathsf{bad}_d]^\eta \\
\mathsf{rhs}_{\mathbb{A}}(\mathsf{bad}_{d+1}) &= [\omega \cdot \mathbf{d} + \mathsf{bad}_d]^\eta.
\end{aligned}$$

The above three properties (a), (b), and (c) can be shown by induction on the layer. For layer L_1 all three properties are trivially true. Now, consider layer L_{d+1}. Property (a) follows directly from the induction hypothesis for layer L_d. Since the linear orders $\mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)$ and $\mathsf{val}_{\mathbb{A}}(\mathsf{bad}_d)$ are shuffles, (c) holds for layer L_{d+1} too. Finally, for (b) we consider two cases:

Case 1. $v \in L_{d+1}$ is an and-gate. Let $v_1, v_2 \in L_d$ be the inputs for v. First, assume that v evaluates to true. Then, v_1 and v_2 both evaluate to true. Hence, by induction, we get $\mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_1}) \cong \mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_2}) \cong \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)$. Thus,

$$\begin{aligned}
\mathsf{val}_{\mathbb{A}}(\mathsf{test}_v) &= [\omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_1}), \omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_2}), \omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)]^\eta \\
&\cong [\omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)]^\eta \\
&= \mathsf{val}_{\mathbb{A}}(\mathsf{good}_{d+1}).
\end{aligned}$$

For the other direction assume that

$$\begin{aligned}
\mathsf{val}_{\mathbb{A}}(\mathsf{test}_v) &= [\omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_1}), \omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_2}), \omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)]^\eta \\
&\cong [\omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)]^\eta.
\end{aligned}$$

Since neither $\mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_1})$ nor $\mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_2})$ nor $\mathsf{val}_{\mathbb{A}}(\mathsf{good}_d)$ contains an interval isomorphic to $\omega \cdot \mathbf{d}$, [22, Lemma 23] implies that

$$\omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_1}) \cong \omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_2}) \cong \omega \cdot \mathbf{d} + \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d).$$

This implies

$$\mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_1}) \cong \mathsf{val}_{\mathbb{A}}(\mathsf{test}_{v_2}) \cong \mathsf{val}_{\mathbb{A}}(\mathsf{good}_d).$$

Finally, the induction hypothesis yields that both v_1 and v_2, and hence also v evaluate to true.

Case 2. $v \in L_{d+1}$ is an or-gate. We can use similar arguments as for Case 1. □

We do not know, whether the lower bound from Theorem 53 holds for ordinary expressions too (instead of succinct expressions).

Theorem 54. *The following problem is* P*-hard (and hence* P*-complete):*

INPUT: Two DFAs $\mathcal{A}_1$ *and* $\mathcal{A}_2$*.*
QUESTION: $(L(\mathcal{A}_1); \leq_{\mathsf{lex}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{lex}})$*?*

Proof. Note that by Theorem 18 the problem belongs to P. For P-hardness, it suffices by Theorem 53 to construct in logspace from a given succinct expression $\mathbb{A}$ (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 [36, Proposition 2]. □

Theorem 18 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 55. *The following problem is* PSPACE*-hard:*

INPUT: Two NFAs $\mathcal{A}_1$ *and* $\mathcal{A}_2$*.*
QUESTION: $(L(\mathcal{A}_1); \leq_{\mathsf{lex}}) \cong (L(\mathcal{A}_2); \leq_{\mathsf{lex}})$*?*

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 $\{a, b\}^*$ [35]. 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 NFAs for the following languages:

$$\begin{aligned} L_1 &= \{a,b\}^* b\, \$_1 \\ L_2 &= K\, b\, \{0,1\}^* 1 \\ L_3 &= \{a,b\}^* b\, \$_2 \\ L &= L_1 \cup L_2 \cup L_3 \end{aligned} \tag{12}$$

It follows that

$$(L; \leq_{\mathsf{lex}}) \cong \sum_{w \in \{a,b\}^* b} \mathcal{L}(w),$$

(the sum is taken over all words from $\{a,b\}^* b$ in lexicographic order), where for all $u \in \{a,b\}^*$:

$$\mathcal{L}(ub) \cong \begin{cases} \mathbf{1} + \eta + \mathbf{1} & \text{if } u \in K \\ \mathbf{2} & \text{else.} \end{cases}$$

Hence, if $K \neq \{a,b\}^*$, then $(L; \leq_{\mathsf{lex}})$ contains an interval isomorphic to $\mathbf{2}$ and therefore is not dense. Hence $(L; \leq_{\mathsf{lex}}) \not\cong \eta$. 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. $\square$

Remark 56. *The proof of Theorem 55 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 is* PSPACE*-complete, since we can check in polynomial space whether $(L(\mathcal{A}); \leq_{\mathsf{lex}}) \cong \eta$: In polynomial time, we can construct an NFA $\mathcal{B}$ that accepts a convolution of two words*[8] *$u \otimes v$ if and only if $u, v \in L(\mathcal{A})$ and there exist words $w_1, w_2, w_3 \in L(\mathcal{A})$ such that $w_1 <_{\mathsf{lex}} u <_{\mathsf{lex}} w_2$ and ($v \leq_{\mathsf{lex}} u$ or $u <_{\mathsf{lex}} w_3 <_{\mathsf{lex}} v$). Then, $(L(\mathcal{A}); \leq_{\mathsf{lex}}) \cong \eta$ if and only if $\mathcal{B}$ accepts the set of all convolutions $u \otimes v$ with $u, v \in L(\mathcal{A})$. The latter can be checked in polynomial space.*

Remark 57. *In [10] it is shown that the problem, whether for a given context-free language L the linear order $(L; \leq_{\mathsf{lex}})$ is isomorphic to η, is undecidable.*

[8] The convolution of the words $a_1 a_2 \cdots a_m$ and $b_1 b_2 \cdots b_n$ is the word $(a_1, b_1)(a_2, b_2) \cdots (a_k, b_k)$, where $k = \max\{m, n\}$, $a_i = \#$ (a dummy symbol) for $m < i \leq k$ and $b_i = \#$ for $n < i \leq k$.

This result is shown by a reduction from Post's correspondence problem. Note that this result can be also easily deduced using the technique from the above proof: If we start with a pushdown automaton for $\mathcal{A}$ instead of an NFA, then the language L from (12) *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 a DFA (resp., NFA). To complete the picture, we will finally consider the isomorphism problem for linear orders that consist of a lexicographically ordered finite language, where the latter is represented by a 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.

Proposition 58. *It is* $\mathsf{C_=L}$*-complete (resp.* $\mathsf{C_=P}$*-complete) to check whether two given acyclic DFAs (resp., acyclic NFAs) accept languages of the same size.*

Proof. The upper bounds are easy: There exists a nondeterministic polynomial time (resp., logspace) machine, which gets an NFA (resp. a DFA) $\mathcal{A}$ over an alphabet Σ as input, and has precisely $|L(\mathcal{A})|$ many accepting paths. Let n be the number of states of n. The machine first branches nondeterministically for at most $n \cdot \log(|\Sigma|)$ steps and thereby produces a word $w \in \Sigma^{\leq n}$. Then it checks whether $w \in L(\mathcal{A})$ and only accepts it this holds. The checking step can be done in deterministic polynomial time for an NFA and in deterministic logspace for a DFA.

For the lower bound, we first consider the DFA-case. Given two nondeterministic logspace machines M_1, M_2 (over the same input alphabet) together with an input w we can produce in logspace the configuration graphs G_1 and G_2 of M_1 and M_2, respectively, on input w. W.l.o.g. we can assume that G_1 and G_2 are acyclic (one can add a step counter to M_i). Now, from G_i it is straightforward to construct an acyclic DFA $\mathcal{A}_i$ such that $|L(\mathcal{A}_i)|$ is the number of paths in G_i from the initial configuration to the (w.l.o.g. unique) accepting configuration. The latter number is the number of accepting computations of M_i on input w.

Finally, $\mathsf{C_{=}P}$-hardness for NFAs follows from [19, Theorem 2.1], where it was shown that counting the number of words accepted by an NFA is $\#\mathsf{P}$-complete. $\square$

4.9. Ordered trees

Let us briefly discuss the isomorphism problem for ordered regular trees, i.e., regular trees, where the children of a node are linearly ordered. An ordered tree can be viewed as a triple $(A; \leq, R)$, where $(A; \leq)$ is a tree as defined in Section 2.3 and the binary relation R is the disjoint union of relations R_a $(a \in A)$, where R_a is a linear order on the children of a. Now, assume that $L \subseteq \Sigma^*$ is a language with $\varepsilon \in L$ and let $\leq_\Sigma$ be a linear order on Σ. Then, we can define a finitely branching ordered regular tree $\mathsf{oT}(L, \leq_\Sigma)$ as follows:

$$\mathsf{oT}(L, \leq_\Sigma) = (L; \leq_{\mathsf{pref}}, \textstyle\bigcup_{u \in L} R_u),$$

where R_u is the relation

$$R_u = \{(v, w) \mid v, w \text{ are children of } u \text{ in } (L; \leq_{\mathsf{pref}}), v \leq_{\mathsf{lex}} w\}.$$

This means that we order the children of a node $u \in L$ lexicographically. If $\mathcal{A}$ is a (deterministic or nondeterministic) finite automaton for L, then we simply write $\mathsf{oT}(\mathcal{A}, \leq_\Sigma)$ for $\mathsf{oT}(L(\mathcal{A}), \leq_\Sigma)$. In the following, we will omit the order $\leq_\Sigma$ on the alphabet. The proof of the following result combines ideas from the proof of Theorem 5 with Theorem 18.

Proposition 59. *The following problem is* P*-complete:*

INPUT: Two DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ with $\varepsilon \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.
QUESTION: $\mathsf{oT}(\mathcal{A}_1) \cong \mathsf{oT}(\mathcal{A}_2)$?

Proof. The lower bound follows easily from Theorem 54. Given a DFA $\mathcal{A}$ for a language $L \subseteq \Sigma^*$, we add a new smallest symbol # to Σ. Then, the language $L\#$ is prefix-free (no word is prefix of another word), $(L; \leq_{\mathsf{lex}}) \cong (L\#; \leq_{\mathsf{lex}})$, and a DFA for $L\#$ can be easily constructed from the DFA $\mathcal{A}$. Moreover, for two languages $L_1, L_2 \subseteq \Sigma^*$ we have $(L_1\#; \leq_{\mathsf{lex}}) \cong (L_2\#; \leq_{\mathsf{lex}})$ if and only if $\mathsf{oT}(L_1\# \cup \{\varepsilon\}) \cong \mathsf{oT}(L_2\# \cup \{\varepsilon\})$.

For the upper bound, it suffices (similarly to the proof of Theorem 5) to take a DFA $\mathcal{A} = (Q, \Sigma, \delta, F)$ without initial state and two states $p, q \in F$, and to check in polynomial time, whether $\mathsf{oT}(\mathcal{A}, p) \cong \mathsf{oT}(\mathcal{A}, q)$, where $\mathsf{oT}(\mathcal{A}, r) = \mathsf{oT}(Q, \Sigma, \delta, r, F)$ for $r \in F$. Define the following equivalence relation on F:

$$\mathsf{iso} = \{(p, q) \in F \times F \mid \mathsf{oT}(\mathcal{A}, p) \cong \mathsf{oT}(\mathcal{A}, q)\}.$$

We show that iso can be computed in polynomial time. As in the proof of Theorem 5, this will be done with a partition refinement algorithm. We need a few definitions.

Recall from the proof of Theorem 5 the definition of the languages $L(\mathcal{A}, p, C)$ and $K(\mathcal{A}, p, C) \subseteq L(\mathcal{A}, p, C)$ for $p \in F$ and $C \subseteq F$. Assume that R is an equivalence relation on F and let m be the number of equivalence classes of R. Fix an arbitrary bijection f between the alphabet $\{1, \ldots, m\}$ and the set of equivalence classes of R. With R and $p \in F$ we associate a partitioned DFA $\mathcal{A}(p, R)$ as follows: Take the DFA for the language $K(\mathcal{A}, p, F)$ as defined in the proof of Theorem 5 and set $F_i = f(i)$ $(1 \leq i \leq m)$, which is the set of final states associated with symbol i. Finally, define the regular word $w(p, R) = w(\mathcal{A}(p, R))$ over the alphabet $\{1, \ldots, m\}$. We define the new equivalence relation $\widetilde{R}$ on F as follows:

$$\widetilde{R} = \{(p, q) \in R \mid w(p, R) \cong w(q, R)\}.$$

Thus, $\widetilde{R}$ is a refinement of R which, by Theorem 18, can be computed in polynomial time from R. Let us define a sequence of equivalence relations $R_0, R_1, \ldots$ on F as follows: $R_0 = F \times F$, $R_{i+1} = \widetilde{R_i}$. Then, there exists $k < |F|$ such that $R_k = R_{k+1}$. We claim that $R_k = \mathsf{iso}$.

For the inclusion $\mathsf{iso} \subseteq R_k$, one shows, by induction on i, that $\mathsf{iso} \subseteq R_i$ for all $1 \leq i \leq k$. The point is that for every equivalence relation R on F with $\mathsf{iso} \subseteq R$, we also have $\mathsf{iso} \subseteq \widetilde{R}$. To see this, assume that $\mathsf{iso} \subseteq R$ but there is $(p, q) \in \mathsf{iso}$, which does not belong to $\widetilde{R}$. Since (p, q) belongs to R, we must have $w(p, R) \not\cong w(q, R)$. On the other hand, since $(p, q) \in \mathsf{iso}$, it follows that the regular words $w(p, \mathsf{iso})$ and $w(q, \mathsf{iso})$ are isomorphic. But since $\mathsf{iso} \subseteq R$, $w(p, R)$ is a homomorphic image of $w(p, \mathsf{iso})$ and similarly for $w(q, R)$. Thus, also $w(p, R)$ and $w(q, R)$ are isomorphic, which is a contradiction.

For the inclusion $R_k \subseteq \mathsf{iso}$, we show that if R is an equivalence relation on F such that $R = \widetilde{R}$ (this holds for R_k), then $R \subseteq \mathsf{iso}$. For this, take a pair $(p_1, p_2) \in R$. Take the tree $\mathsf{oT}(\mathcal{A}, p_i)$. We assign types in form of final states to the nodes of $\mathsf{oT}(\mathcal{A}, p_i)$ in the same way as in the proof of Theorem 5. We now construct an isomorphism $f : \mathsf{oT}(\mathcal{A}, p_1) \to \mathsf{oT}(\mathcal{A}, p_2)$ as the limit of isomorphisms f_n, $n \geq 1$. Here, f_n is an isomorphism between the trees that result from $\mathsf{oT}(\mathcal{A}, p_1)$ and $\mathsf{oT}(\mathcal{A}, p_2)$ by cutting off all nodes below level n. Let us call these trees $\mathsf{oT}(\mathcal{A}, p_i)\restriction_n$ ($i \in \{1, 2\}$). Moreover, if an f_n maps a node u_1 of type q_1 to a node u_2 of type q_2, then we will have $(q_1, q_2) \in R$. Assume that f_n is already constructed and let u_1 of type q_1 be a leaf of $\mathsf{oT}(\mathcal{A}, p_1)\restriction_n$. Let $u_2 = f(u_1)$ be of type q_2. Then we have $(q_1, q_2) \in R$ and hence the regular words $w(q_1, R)$ and $w(q_2, R)$ are isomorphic. Let g be an isomorphism. The elements of these regular words correspond to the children of u_1 and u_2, respectively. More precisely, if v_i belongs to the domain of $w(q_i, R)$, then the word $u_i v_i$ is a child of u_i and vice versa. Clearly, g can be also viewed as an isomorphism between the lexicographical orderings on the children of u_1 and u_2, respectively. Moreover, by definition of the regular words $w(q_1, R)$ and $w(q_2, R)$, if g maps some $u_1 v_1$ of type r_1 to $u_2 v_2$ of type r_2, then $(r_1, r_2) \in R$. By choosing such an isomorphism g for every pair $(u_1, f(u_1))$ of leaves in $\mathsf{oT}(\mathcal{A}, p_1)\restriction_n$ and $\mathsf{oT}(\mathcal{A}, p_2)\restriction_n$, respectively, we can extend f_n to f_{n+1}. □

Let us now consider prefix-closed automata. Here, we can improve the upper bound from Theorem 59 to NL.

Proposition 60. *The following problem is* NL*-complete:*

INPUT: Two prefix-closed DFAs $\mathcal{A}_1$ *and* $\mathcal{A}_2$*.*
QUESTION: $\mathsf{oT}(\mathcal{A}_1) \cong \mathsf{oT}(\mathcal{A}_2)$*?*

Proof. Again, it suffices to take a prefix-closed DFA $\mathcal{A} = (Q, \Sigma, \delta, Q)$ without initial state, and two states $p, q \in Q$, and two check in NL, whether

$$\mathsf{oT}(Q, \Sigma, \delta, p, Q) \cong \mathsf{oT}(Q, \Sigma, \delta, q, Q).$$

By the complement closure of NL, it suffices to check nondeterministically in logarithmic space, whether $\mathsf{oT}(Q, \Sigma, \delta, p, Q) \not\cong \mathsf{oT}(Q, \Sigma, \delta, q, Q)$ This can be done as follows: Let $a_1 < a_2 \cdots < a_m$ and $b_1 < b_2 < \cdots < b_n$ the transition labels of the outgoing transitions of p and q, respectively. If $m \neq n$ then clearly $\mathsf{oT}(Q, \Sigma, \delta, p, Q) \not\cong \mathsf{oT}(Q, \Sigma, \delta, q, Q)$ and the algorithm can accept. If $n = m$, then $\mathsf{oT}(Q, \Sigma, \delta, p, Q) \not\cong \mathsf{oT}(Q, \Sigma, \delta, q, Q)$ if and only if there exists $1 \leq i \leq m$ such that $\mathsf{oT}(Q, \Sigma, \delta, \delta(p, a_i), Q) \not\cong \mathsf{oT}(Q, \Sigma, \delta, \delta(q, b_i), Q)$. Hence, the algorithm will simply guess $1 \leq i \leq m$ and replace the state pair (p, q) by $(\delta(p, a_i), \delta(q, b_i))$. In this way, the algorithm only has to store two states of $\mathcal{A}$, which is possible in logspace.

NL-hardness can be shown by a reduction from the complement of the graph accessibility problem. Take a directed graph $G = (V, E)$ and two nodes

	DFA	NFA
acyclic	P-complete	PSPACE-complete
arbitrary		EXPTIME-complete

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

$s, t \in V$. Add to each node of V loops, so that every node $v \in V \setminus \{t\}$ has outdegree n (where n can be taken as the maximal outdegree of a node of G) and t has outdegree $n+1$. Then label the edges of the resulting multigraph arbitrarily by symbols so that we obtain a DFA $\mathcal{A}$ (the initial state is s and all states are final). Then there is no path from s to t in G if and only if the tree $\mathsf{oT}(\mathcal{A})$ is a full n-ary tree. □

Corollary 61. *The following problem is* PSPACE*-complete:*

INPUT: Two prefix-closed NFAs $\mathcal{A}_1$ *and* $\mathcal{A}_2$*.*
QUESTION: $\mathsf{oT}(\mathcal{A}_1) \cong \mathsf{oT}(\mathcal{A}_2)$*?*

Proof. The PSPACE upper bound follows from Proposition 60, using Lemma 1 and the obvious fact that the power set automaton of a given NFA can be produced by a PSPACE-transducer. For the PSPACE lower bound, note that for an NFA $\mathcal{A}$ over an alphabet Σ we have $L(\mathcal{A}) = \Sigma^*$ if and only if $\mathsf{oT}(\mathcal{A})$ is a full $|\Sigma|$-ary tree. But universality for NFAs is PSPACE-complete [35]. □

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 EXPTIME and is PSPACE-hard. Another interesting problem concerns the equivalence problem for straight-line programs (i.e., succinct expressions that generate finite words, or equivalently, acyclic partitioned DFAs, or equivalently, context-free grammars that generate a single word). This problem can be solved in polynomial time [15, 28, 31]. Recall

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

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

that this result is fundamental for our polynomial time algorithm for succinct expressions (Theorem 35). In [11], it was conjectured that the equivalence problem for straight-line programs is P-complete, but this is still open.

Acknowledgments.. Both authors were partially supported by the DFG-project LO 748/8 (Algorithmen für komprimierte Daten). The authors also want to thank Dietrich Kuske for many useful comments.

References

[1] Vince Bárány, Erich Grädel, and Sasha Rubin. Automata-based presentations of infinite structures. In *Finite and Algorithmic Model Theory*, number 379 in London Mathematical Society Lecture Notes Series, pages 1–76. Cambridge University Press, 2011.

[2] Stephen L. Bloom and Zoltán Ésik. Deciding whether the frontier of a regular tree is scattered. *Fundamenta Informaticae*, 55(1):1–21, 2003.

[3] Stephen L. Bloom and Zoltán Ésik. The equational theory of regular words. *Information and Computation*, 197(1-2):55–89, 2005.

[4] Stephen L. Bloom and Zoltán Ésik. Algebraic linear orderings. *International Journal of Foundations of Computer Science*, 22(2):491–515, 2011.

[5] Ronald V. Book and Friedrich Otto. *String–Rewriting Systems.* Springer, 1993.

[6] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[7] Bruno Courcelle. Frontiers of infinite trees. *Informatique Théorique et Applications*, 12(4), 1978.

[8] Bruno Courcelle. The definability of equational graphs in monadic second-order logic. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP 1989)*, number 372 in Lecture Notes in Computer Science, pages 207–221. Springer, 1989.

[9] Zoltán Ésik. Representing small ordinals by finite automata. In *Proceedings of the 12th Annual Workshop on Descriptional Complexity of Formal Systems (DCFS 2010)*, number 31 of *EPTCS*, pages 78–87, 2010.

[10] Zoltán Ésik. An undecidable property of context-free linear orders. *Information Processing Letters*, 111(3):107–109, 2011.

[11] Leszek Gasieniec, Alan Gibbons, and Wojciech Rytter. Efficiency of fast parallel pattern searching in highly compressed texts. In *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science (MFCS'99)*, number 1672 in Lecture Notes in Computer Science, pages 48–58. Springer, 1999.

[12] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. Efficient algorithms for Lempel-Ziv encoding (extended abstract). In *Proceedings of the 5th Scandinavian Workshop on Algorithm Theory (SWAT 1996)*, number 1097 in Lecture Notes in Computer Science, pages 392–403. Springer, 1996.

[13] Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–99, 1977.

[14] Stephan Heilbrunner. An algorithm for the solution of fixed-point equations for infinite words. *Informatique Théorique et Applications*, 14(2):131–141, 1980.

[15] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1&2):143–159, 1996.

[16] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, 66(3):549–566, 2003.

[17] Artur Jez. Faster fully compressed pattern matching by recompression. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP 2012)*, number 7391 in Lecture Notes in Computer Science, pages 533–544. Springer, 2012.

[18] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1), 1990.

[19] Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 551–557, ACM/SIAM, 1995.

[20] Bakhadyr Khoussainov, André Nies, Sasha Rubin, and Frank Stephan. Automatic structures: richness and limitations. *Logical Methods in Computer Science*, 3(2):2:2, 18 pp. (electronic), 2007.

[21] Dietrich Kuske. Isomorphisms of scattered automatic linear orders. In *Proceedings of the 21st Annual Conference of the EACSL (CSL 2012)*, number 14 of LIPIcs, pages 455–469, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.

[22] Dietrich Kuske, Jiamou Liu, and Markus Lohrey. The isomorphism problem on classes of automatic structures. Technical report, arXiv.org, 2010. `http://arxiv.org/abs/1001.2086`.

[23] Dietrich Kuske, Jiamou Liu, and Markus Lohrey. The isomorphism problem on classes of automatic structures with transitive relations. to appear in *Transactions of the American Mathematical Society*, 2012.

[24] Yury Lifshits. Processing compressed texts: A tractability border. In *Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM 2007)*, number 4580 in Lecture Notes in Computer Science, pages 228–240. Springer, 2007.

[25] Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC'92)*, pages 400–404. ACM Press, 1992.

[26] Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

[27] Markus Lohrey and Christian Mathissen. Isomorphism of regular trees and words. In *Proceeding of the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011)*, number 6756 in Lecture Notes in Computer Science, pages 210–221. Springer, 2011.

[28] Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997.

[29] Masamichi Miyazaki, Ayumi Shinohara, and Masayuki Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching (CPM 97)*, number 1264 in Lecture Notes in Computer Science, pages 1–11. Springer, 1997.

[30] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

[31] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *Proceeding of the 2nd Annual European Symposium on Algorithms (ESA'94)*, number 855 in Lecture Notes in Computer Science, pages 460–470. Springer, 1994.

[32] Wojciech Plandowski and Wojciech Rytter. Complexity of language recognition problems for compressed words. In Juhani Karhumäki, Hermann A. Maurer, Gheorghe Paun, and Grzegorz Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 262–272. Springer, 1999.

[33] J. Rosenstein. *Linear Ordering.* Academic Press, 1982.

[34] Wojciech Rytter. Grammar compression, LZ-encodings, and string algorithms with implicit input. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, number 3142 in Lecture Notes in Computer Science, pages 15–27. Springer, 2004.

[35] Larry J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the 5th Annual*

ACM Symposium on Theory of Computing (STOCS'73), pages 1–9. ACM Press, 1973.

[36] Wolfgang Thomas. On frontiers of regular trees. *Informatique Théorique et Applications*, 20(4):371–381, 1986.