RAIRO. INFORMATIQUE THÉORIQUE ET APPLICATIONS

KAI SALOMAA
DERICK WOOD
SHENG YU

Complexity of E0L structural equivalence

RAIRO. Informatique théorique et applications, tome 29, nº 6 (1995), p. 471-485

<http://www.numdam.org/item?id=ITA_1995__29_6_471_0>

© AFCET, 1995, tous droits réservés.

L'accès aux archives de la revue « RAIRO. Informatique théorique et applications » implique l'accord avec les conditions générales d'utilisation (http://www.numdam.org/conditions). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques
http://www.numdam.org/

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 29, n° 6, 1995, p. 471 à 485)

COMPLEXITY OF E0L STRUCTURAL EQUIVALENCE *

by Kai SALOMAA ([1]), Derick WOOD ([2]) and SHENG YU ([2])

Communicated by Jean BERSTEL

Abstract. – *We show that the E0L structural equivalence problem is logspace hard for deterministic exponential time. Also, we show that this question can be solved in linear space by a synchronized alternating Turing machine, and thus establish an exponential space upper bound for its complexity. The equivalence of finite tree automata is shown to be logspace reducible to context-free structural equivalence. The converse reduction is well known and thus context-free structural equivalence is complete for deterministic exponential time.*

Résumé. – *Nous prouvons que l'équivalence structurelle des E0L-systèmes est difficile en espace logarithmique et temps déterministe exponentiel. Nous montrons également que cette question peut être résolue en espace linéaire par une machine de Turing alternante, ce qui établit une borne supérieure exponentielle en place pour sa complexité. On prouve que l'équivalence d'automates finies d'arbres est réductible en place logarithmique à l'équivalence structurelle des langages algébriques. La réduction réciproque est bien connue, et ainsi l'équivalence structurelle des langages algébrique est complète pour le temps exponentiel déterministe.*

1. INTRODUCTION

Two context-free grammars are said to be structurally equivalent if the corresponding sets of syntax trees are equal when we disregard the nonterminals labeling the internal nodes of the trees. While it is well known that language equivalence of context-free grammars is undecidable, it was shown by McNaughton [10], and, Paull and Unger [14] that structural equivalence can be decided effectively.

The notion of structural equivalence can be defined analogously for context independent L grammars (E0L grammars). The structural equivalence problem of E0L grammars was first considered by Ottmann and Wood

(*) Received november 1994, accepted May 1995.
Research supported by Natural Sciences and Engineering Research Council of Canada grants.
([1]) Department of Mathematics, University of Turku, 20500 Turku, Finland.
([2]) Department of Computer Science, University of Western Ontario, London, Ontario, Canada N6A 5B7.

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/95/06/$ 4.00/© AFCET-Gauthier-Villars

[12, 13]. Two different decidability proofs for this problem are given in [11, 16].

Contrasting the results for context-free (CF) and E0L grammars it was shown recently [17] that the structural equivalence problem for ET0L (tabled E0L) grammars is undecidable. Two stronger notions of equivalence, syntax equivalence [17] and strong structural equivalence [7], have been shown to be decidable also for ET0L grammars. Two grammars are syntax equivalent if the sets of syntax trees of the grammars are equal. Strong structural equivalence is a restricted form of structural equivalence where additionally one requires that the corresponding syntax trees use the same sequence of tables. For E0L grammars the notion of strong structural equivalence naturally reduces to structural equivalence.

In view of the undecidability result for ET0L grammars it could be expected that also E0L structural equivalence should be computationnaly difficult. Here we establish that the CF and E0L structural equivalence problems are in fact provably difficult: both are hard for deterministic exponential time with respect to logarithmic space many-one reductions.

It is well known that context-free syntax trees can be recognized by finite tree automatons. Conversely, the family of sets of syntax trees clearly does not contain all recognizable tree languages. Context-free grammars define as their syntax trees exactly the local (or locally testable) tree languages [4]. Also, here we are considering the structures of syntax trees where the internal nodes are unlabeled. However, by adding new nodes we can construct a syntax tree whose structure codes a successful computation of the tree automaton. In this way, for arbitrary given finite tree automata $\mathcal{A}_i$, $i = 1, 2$, we can (in logspace) construct CF grammars that are structurally equivalent if and only if the tree automata $\mathcal{A}_1$ and $\mathcal{A}_2$ recognize the same tree language. The equivalence of finite tree automata is logspace complete for deterministic exponential time [18], and thus we have a completeness result also for CF structural equivalence.

It is to be expected that E0L structural equivalence has at least the same complexity as CF structural equivalence, and, in fact, it is not difficult to see that E0L structural equivalence is logspace hard for deterministic exponential time. However, the decidability proofs of [11, 16] yield only a multiple exponential time upper bound. These results use as their starting point invertible grammars structurally equivalent to the original grammars. The construction of a structurally equivalent invertible grammar [13] already causes an exponential blow-up in the number of nonterminals. Here we

present a more efficient decision algorithm. We show that E0L structural equivalence can be decided by a linear space bounded synchronized alternating Turing machine. Our algorithm is inspired by the constructions in [18]. The results of [2, 5, 23] then yield a deterministic exponential space upper bound for the complexity of E0L structural equivalence. The same upper bound is shown to hold also for the complexity of strong structural equivalence of ET0L grammars.

2. PRELIMINARIES

Here we recall briefly some notation and definitions concerning trees and tree automata. We assume that the reader is familiar with central notions of formal language theory, see [24], and L-systems in particular, see [15]. More details about tree automata can be found in [4]. For unexplained notions of complexity theory we refer the reader to [1]. Complexity issues of L grammars are considered e.g. in [8, 9, 21, 22].

The set of finite words over an alphabet A is denoted A^* and λ is the empty word. Also, $A^+ = A^* - \{\lambda\}$. The length of $w \in A^*$ is $|w|$. The cardinality of a finite set A is denoted by $\#A$ and the power set of A is $\mathcal{P}(A)$. N_0 is the set of nonnegative integers.

Symbols Σ and Ω stand always for finite raked alphabets. The rank of $\sigma \in \Sigma$ is denoted $\mathrm{rank}_\Sigma(\sigma)$ and $\Sigma_m = \{\sigma \in \Sigma | \mathrm{rank}_\Sigma(\sigma) = m\}$, $m \geq 0$. The set of ($\Sigma-$)trees over a ranked alphabet Σ is F_Σ. If X is a set of auxiliary symbols, the set $F_\Sigma(X)$ consists of Σ-trees where some leaves are labeled by elements of X. Elements of $F_\Sigma(X)$ are called ΣX-trees. Σ-trees can be seen in the natural way as labeled tree graphs where each node having m sons is labeled by an element of rank m. When needed, we associate to a ΣX-tree t a corresponding set of nodes $\mathrm{dom}(t)$ (the domain of t) and view the tree t as a mapping $\mathrm{dom}(t) \to \Sigma \cup X$. We assume that notions such as the height, the root, a leaf and a subtree of a tree are well known. The height of $t \in F_\Sigma$ is denoted $\mathrm{hg}(t)$. For $t \in F_\Sigma(X)$ and $u \in \mathrm{dom}(t)$, $t(u)$ denotes the label of the node u and the subtree at node u is t/u. For $x_i \in X$ and $t_i \in F_\Sigma(X)$, $i = 1, \ldots, m$, $t(x_1 \leftarrow t_1, \ldots, x_m \leftarrow t_m)$ denotes the tree obtained from t by replacing all leaves labeled by x_i with the tree t_i, $i = 1, \ldots, m$.

Let Σ and Ω be ranked alphabets and let $n \in N_0$ be such that $\Sigma_i = \emptyset$ for all $i > n$. Assume that for each $i \in \{0, \ldots, n\}$ we are given a mapping $h_i : \Sigma_i \to F_\Omega(X_i)$, $X_i = \{x_1, \ldots, x_i\}$. The mappings h_i, $i = 0, \ldots, n$,

determine the *tree homomorphism* defined inductively by the condition: If $m \geq 0$, $\sigma \in \Sigma_m$, and $t_1, \ldots, t_m \in F_\Sigma$ then

$$h(\sigma(t_1, \ldots, t_m)) = h_m(\sigma)(x_1 \leftarrow h(t_1), \ldots, x_m \leftarrow h(t_m)).$$

A (nondeterministic) *finite tree automaton* FTA is a four-tuple $\mathcal{A} = (\Sigma, A, A', \delta)$, where Σ is a ranked alphabet of input symbols, A is a finite set of states, $A' \subseteq A$ is the set of final states, and the state transition relation δ defines for every $\sigma \in \Sigma_m$, $m \geq 0$, a mapping $\sigma_\delta : A^m \to \mathcal{P}(A)$. The state transition relation associates in the well known way to each tree $t \in F_\Sigma$ the set of states t_δ that $\mathcal{A}$ can reach at the root of t. The *tree language* recognized by the automaton $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in F_\Sigma : t_\delta \cap A' \neq \emptyset\}$.

We assume that the reader is familiar with the notion of space bounded (many-one) reducibility [1]. The deterministic exponential time and exponential space complexity classes are denoted as follows [1]:

$$DEXT = \bigcup_{c \in \mathcal{N}_0} DTIME(2^{c \cdot n}), \quad EXPSPACE = \bigcup_{c \in \mathcal{N}_0} DSPACE(2^{c \cdot n}).$$

Our algorithm for the E0L structural equivalence problem uses *synchronized alternating Turing machines*, SATM's. Below we give a short description of this machine model. A more detailed formal definition of SATM's can be found in [2, 5, 6, 19, 23]. the definition of alternating Turing machines appears, for instance, in [3] or [1] Vol. II.

An SATM is an alternating Turing machine **M** where some internal states contain a second component from a finite set S of synchronizing symbols; these are called synchronizing states. As usual for alternating machines, the states of **M** are partitioned into existential and universal states. A configuration is existential (universal) if the corresponding state of **M** is existential (universal).

We denote by comp (**M**) the set of computation trees of **M** where **M** is viewed as an ordinary alternating machine. A tree $T \in \text{comp}(\mathbf{M})$ is constructed by taking all descendants of universal configurations and exactly one descendant of existential configurations. The *synchronizing sequence* of a node u of T is the sequence of synchronizing symbols ($\in S$) occurring in the configurations labeling the nodes on the path from the root of T to u.

A *synchronized computation tree* of the STAM **M** is a tree $T \in \text{comp}(\mathbf{M})$ satisfying the following condition: for arbitrary nodes u_1 and u_2 of T the

synchronizing sequence of u_1 is a prefix of the synchronizing sequence of u_2 or *vice versa*.

The language accepted by the STAM **M** consists of all input words w such that there exists a finite synchronized computation tree of **M** where the root is labeled by the initial configuration corresponding to w and leaf is labeled by an accepting configuration. Intuitively, the synchronization condition can be interpreted as follows: when one parallel branch of an alternating computation enters a synchronizing state it must wait until all other branches either halt of enter a synchronizing state having the same synchronizing symbol.

The class of languages accepted by $s(n)$ space bounded synchronized alternating machines is denoted $SASPACE(s(n))$. It is established in [5] (see also [2, 23]) that for a space-constructible function $s(n) \geq \log n$

$$SASPACE(s(n)) = \bigcup_{c \in N_0} DSPACE(c^{s(n)}). \tag{1}$$

3. STRUCTURAL EQUIVALENCE

We recall the definition of the notion of structural equivalence of sequential and parallel context-free type grammars [10, 13, 16]. We define the structure trees as Σ-trees where the ranked alphabet Σ has exactly one symbol of each rank $m \geq 1$. In the more usual definition the internal nodes have no labels, but the definitions are clearly equivalent since a node of an unlabeled tree implicitly contains the information about the number of immediate successors. Viewing structure trees as Σ-trees allows just a convenient characterization for them using tree automata.

A context-free (CF) or an E0L grammar is defined as a four-tuple

$$G = (V, T, S, P), \tag{2}$$

where V is a finite nonterminal alphabet, T is a finite set of terminal symbols, $S \in V$ is the initial nonterminal, and P is a finite set of productions of the form $X \to w$, where $X \in V$ and $\omega \in (V \cup T)^*$. (We do not allow rewriting of terminals.) The derivations of an E0L grammar are performed in parallel, that is, at each step all symbols of a sentential form are rewritten, whereas the CF derivations are performed sequentially. We call a grammar G as

in (2) a CF or E0L grammar depending on which type of derivation relation we have in mind. For $X \in V$ let

$$G[X] = (V, T, X, P). \tag{3}$$

In the following, let G be as in (2). We denote by $F(G)$ the set of all trees (tree graphs) where the nodes are labeled by elements of $V \cup T \cup \{\hat{\lambda}\}$. Here $\hat{\lambda}$ is a new symbol that will be used to denote the erasing productions in a syntax tree. In the following we define the sequential (respectively, parallel) derivation relation $\rightarrow_G^s \subseteq F(G) \times F(G)$ (respectively, $\rightarrow_G^p \subseteq F(G) \times F(G)$).

Let $T, T' \in F(G)$ and $z \in \{s, p\}$. Then $T \rightarrow_G^z T'$ if and only if T' is obtained from T as follows.

(i) $z = s$: Let u be a leaf of T labeled by a nonterminal $X \in V$. Let $X \rightarrow a_1 \ldots a_n$, $n \geq 0$, $a_1, \ldots, a_n \in V \cup T$, be a production of P. If $n \geq 1$, then in T' the node u has n successors labeled, respectively, by the symbols $a_1, \ldots, a_n$. If $n = 0$, i.e., the right side of the production is the empty word, then in T' one attaches for the node u a single successor labeled by the symbol $\hat{\lambda}$.

(ii) $z = p$: Assume that all leaves of T are labeled by elements of $V \cup \{\hat{\lambda}\}$. (If some leaf is labeled by a terminal, the derivation cannot be continued). Then corresponding to each leaf u labeled by a nonterminal X one chooses a production with left side X and attaches new successors for the node u as in (i) above.

The set of *z-syntax trees*, $z \in \{s, p\}$ of the grammar G is defined by

$$S_{[z]}(G) = \{T \in F(G) : S' (\rightarrow_G^z)^* T\},$$

where S' is the tree with a single node labeled by the initial nonterminal S. A syntax tree $T \in S_{[z]}(G)$ is said to be *terminal* if all leaves of T are labeled by elements of $T \cup \{\hat{\lambda}\}$. The set of terminal z-syntax trees of G is denoted $TS_{[z]}(G)$.

The terminal z-syntax trees correspond exactly to the derivations of terminal words, where depending on the value of z we view G as a CF or E0L grammar. The language generated by G consists of the yields of the terminal z-syntaw trees. The yield of a tree is obtained by catenating the symbols labeling the leaves from left to right and replacing the symbol $\hat{\lambda}$ with the empty word. The *structure* of a terminal syntax tree T is, intuitively, the leaf-labeled tree that is obtained from T by removing the nonterminals

labeling the internal nodes. In the formal definition we want to define the structures of syntax trees as Σ-trees for a suitably chosen ranken alphabet Σ.

Let $M_G = \max\{|w| : X \to w \in P\}$. Choose

$$\Sigma^G = \{c_1, \ldots, c_{M_G}\} \cup T \cup \{\hat{\lambda}\}, \tag{4}$$

where $\Sigma_0^G = T \cup \{\hat{\lambda}\}$ and $\mathrm{rank}_{\Sigma^G}(c_i) = i$, $i = 1, \ldots, M_G$. The mapping

$$\mathrm{str}_G : TS_{[z]}(G) \to F_{\Sigma^G}$$

relabels each internal node $u \in \mathrm{dom}(t)$, $T \in TS_{[z]}(G)$, with c_i, where i is the number of immediate successors of the node u. Also, we denote

$$STS_{[z]}(G) = \{\mathrm{str}_G(T) : T \in TS_{[z]}(G)\}, \quad z \in \{s, p\}.$$

Context-free or E0L grammars G_1 and G_2 are said to be *structurally equivalent* if

$$STS_{[z]}(G_1) = STS_{[z]}(G_2), \tag{5}$$

with, respectively, $z = s$ or $z = p$. When considering the computational complexity of the question whether given grammars G_1, $i = 1, 2$, satisfy (5), the size of a grammar G_i, size (G_i), is considered to be the length of an encoding of the grammar G_i over a fixed (binary) alphabet, see e.g. [8].

4. RESULTS

We show that the context-free structural equivalence problem is logspace complete for $DEXT$ by reducing the equivalence of FTA's to it. For E0L grammars, we show that structural equivalence is hard in $DEXT$ with respect to logspace reductions and that structural equivalence can be decided by a synchronized alternating Turing machine [5, 19] in linear space, which implies that the problem is in $EXPSPACE$.

LEMMA 4.1: *Given* $\mathcal{A}_i = (\Sigma, A_i, A_i', \delta_i) \in FTA$, $i = 1, 2$, *we can construct using logarithmic workspace* CF *grammars* G_i, $i = 1, 2$, *such that*

$$L(\mathcal{A}_1) = L(\mathcal{A}_2) \; iff \; STS_{[s]}(G_1) = STS_{[s]}(G_2). \tag{6}$$

Proof: Let $i \in \{1, 2\}$ be fixed. Roughly speaking, the grammar G_i generates as its syntax threes the successful computations of the

automaton $\mathcal{A}_i$. The grammar attaches additional nodes to the syntax tree in order to guarantee that for $t \in TS_{[s]}(G_i)$, $\mathrm{str}_{G_i}(t)$ determines uniquely the underlying input tree ($\in F_\Sigma$) of the corresponding computation of $\mathcal{A}_i$.

Denote $\overline{\Sigma} = \{\overline{\sigma} : \sigma \in \Sigma\}$ and let \$ be a new symbol not belonging to Σ. We define a ranked alphabet Ω by setting $\Omega_0 = \overline{\Sigma}$, $\Omega_1 = \{\$\} \cup (\Sigma_0 \times A_i)$, and $\Omega_{m+1} = \Sigma_m \times A_i$, $m \geq 1$. The symbols of Ω are used as terminals and nonterminals of the grammar G_i and the production of the grammar are defined so that the terminal syntax trees can be seen as well-formed Ω-trees. This allow us to formulate the correspondence between $TS_{[s]}(G_i)$ and the set of computations of $\mathcal{A}_i$ by way of a tree homomorphism.

We define $G_i = (\Omega - \Omega_0, \Omega_0, \$, P_i)$ where P_i consists of the following productions:

(i) $\$ \to (\sigma, a)$ for $\sigma \in \Sigma$, $a \in A_i'$.

(ii) $(\sigma, a) \to \overline{\sigma}(\sigma_1, a_1)\ldots(\sigma_m, a_m)$, if $m \geq 1$, $\sigma \in \Sigma_m$, $\sigma_1, \ldots, \sigma_m \in \Sigma$, $a, a_1, \ldots, a_m \in A_i$, and $a \in \sigma_{\delta_i}(a_1, \ldots, a_m)$.

(iii) $(\sigma, a) \to \overline{\sigma}$ if $\sigma \in \Sigma_0$, $a \in \sigma_{\delta_i}$.

It is clear that $TS_{[s]}(G_i) \subseteq F_\Omega$ *i.e.*, syntax trees of G_i can be interpreted as Ω-trees. We use the nonterminal \$ to begin the derivations since a grammar is allowed to have only one initial nonterminal. We define the tree homomorphism $h : F_\Omega \to F_\Sigma$ by the following conditions:

- $h_1(\$) = x_1$.
- For $(\sigma, a) \in \Omega_{m+1}$, $m \geq 0$, $\sigma \in \Sigma_m$, $a \in A_i$: $h_{m+1}((\sigma, a)) = \sigma(x_2, \ldots, x_{m+1})$.
- For $\omega \in \Omega_0$, $h_0(\omega)$ can be defined arbitrarily.

Intuitively, the tree homomorphism h just deletes that state information and the leftmost subtree of each node.

Claim 1. Let $a \in A_i$ and $t = \sigma(t_1, \ldots, t_m) \in F_\Sigma$, $m \geq 0$. There exists $r \in TS_{[s]}(G_i[(\sigma, a)]) \cap h^{-1}(t)$ iff $a \in t_{\delta_i}$. For the notation see (3).

The claim follows from the fact that the grammar G_i simulates the computation of $\mathcal{A}_i$ in the top-down direction. the proof uses tree induction on t and we leave it for the reader. By Claim 1 and the choice of the rules (i) we have

$$L(\mathcal{A}_i) = h(TS_{[s]}(G_i)). \tag{7}$$

Note that the tree homomorphism h deletes the root node labeled by \$ from the syntax trees.

Without loss of generality, we can assume that $\Sigma^{G_1} = \Sigma^{G_2}$. (The notation is from (4).) We denote this set by Σ^G and define the tree homomorphism $g : F_{\Sigma \to F_{\Sigma^G}}$ by setting

$$g_m(\sigma) = c_{m+1}(\overline{\sigma}, x_1, \ldots, x_m), \ \sigma \in \Sigma_m, \ m \geq 0. \tag{8}$$

Let $\overline{h}$ be the restriction of h to the domain $TS_{[s]}(G_i)\,(\subseteq F_\Omega)$ and let $g_{(i)}$ be the restriction of g to $L(\mathcal{A}_i)$. It is clear that

$$\mathrm{str}_{G_i} \circ \overline{h}^{-1} = g_{(i)}. \tag{9}$$

From the definition (8) it is easy to see that g is injective. Thus by (7) and (9), $g_{(i)}$ is a bijection $L(\mathcal{A}_i) \to STS_{[s]}(G_i)$. Since g is an injective extension of both $g_{(i)}$ and $g_{(2)}$, it follows that (6) holds. Clearly, the grammar G_i, $1 \leq i \leq 2$, can be constructed from $\mathcal{A}_i$ using logarithmic space. □

THEOREM 4.1: *The structural equivalence problem for context-free grammars is logspace complete in DEXT*

Proof: The equivalence of finite tree automata is logspace hard for $DEXT$ [18]. This implies by Lemma 4.1 that CF structural equivalence is logspace hard for $DEXT$. The set of structures of terminal syntax trees of a CF grammar can be recognized by a finite tree automaton [20, 4]. It is easy to see that the automaton can be constructed in linear time. Thus CF structural equivalence is in $DEXT$ by [18]. □

In the following lemma we show that the CF structural equivalence problem can be reduced in logspace to E0L strcutural equivalence.

LEMMA 4.2: *Given CF grammars $G_i = (v_i, T_i, S_i, P_i)$, $i = 1, 2$, we can construct in logspace E0L grammars G'_i, $i = 1, 2$, such that*

$$STS_{[s]}(G_1) = STS_{[s]}(G_2) \ \textit{iff} \ STS_{[p]}(G'_1) = STS_{[p]}(G'_2). \tag{10}$$

Proof: Choose

$$G'_i = (V_i \cup \{\$\}, T_i \cup \{\&\}, S_i, P'_i), \qquad 1 \leq i \leq 2,$$

where \$, & are new symbols and

$$P'_i = P_i \cup \{A \to A\$: A \in V_i\} \cup \{\$ \to \$, \ \$ \to \&\}.$$

In the following, let $i \in \{1, 2\}$ be fixed. Let $t \in STS_{[s]}(G'_i)$ and let $u \in \mathrm{dom}(t)$ be such that $t(u) = \&$. Let v be the closet predecessor of u having more than one sons. Then

$$t/v = c_1(t_1, t_2),$$

where t_2 is a unary tree having the leaf u. The tree obtained from t by *pruning* the unary branch ending at u (u-branch) is defined as

$$\mathrm{prun}_{(\&, u)}(t) = t(v \leftarrow t_1).$$

intuitively, $\mathrm{prun}_{(\&, u)}(t)$ is the structure of the syntax tree that is obtained from t by canceling the production $A \to A\$$ at node v (and continuing the derivation as in the left successor of v). By the &-pruned tree of t, $\mathrm{prun}_{\&}(t)$, we mean the tree obtained from t by pruning every u-branch where $u \in \mathrm{dom}(t)$ is labeled by &. Clearly for $t \in STS_{[p]}(G'_i)$, $\mathrm{prun}_{\&}(t) \in STS_{[s]}(G_i)$, $i = 1, 2$. Conversely, for an arbitrary tree $r \in STS_{[s]}(G_i)$, there exists $r' \in STS_{[p]}(G'_i)$ such that $\mathrm{prun}_{\&}(r') = r$. The underlying syntax tree of r (*i.e.*, any tree in $\mathrm{str}^{-1}_{G_i}(r)$) can be made to be a parallel syntax tree of G'_i by continuing all "too short" branches with rewrite steps of the form $A \to A\$$. The resulting parallel syntax tree yields as its structure r' where $\mathrm{prun}_{\&}(r') = r$. Thus we have

$$STS_{[s]}(G_i) = \mathrm{prun}_{\&}(STS_{[p]}(G'_i)). \tag{11}$$

We say that a tree t' is obtained from $t \in STS_{[s]}(G_i)$ by *grafting* a &-branch at node $u \in \mathrm{dom}(t)$, if

$$t' = t(u \leftarrow c_2(t/u, c_1^k(\&))), \qquad k \geq 1.$$

Denote $H_i = STS_{[p]}(G'_i)$. The set H_i consists of exactly all trees that are obtained from the trees $\mathrm{prun}_{\&}(H_i)$ by grafting new &-branches in such a way that the distance from the root to every leaf not labeled by $\hat{\lambda}$ is the same. Thus clearly,

$$\mathrm{prun}_{\&}(H_1) = \mathrm{prun}_{\&}(H_2) \quad \text{implies } H_1 = H_2.$$

The above and (11) imply that (10) holds. □

The decision algorithms for E0L structural equivalence given by [11, 16] are extremely inefficient. Both algorithms need as their starting point invertible grammars structurally equivalent to the input grammars, and

the construction of the invertible grammars already causes an exponential increase in the number of nonterminal symbols. Thus, for instance, the algorithm directly following (Lemmas 3.3 and 3.4 of) [16] gives only a triple-exponential time bound. The decision algorithm of [11] yields also a multi-exponential time bound, the complexity of the decision method is discussed on p. 143 of [11]. Furthermore, [11, 16] assume that the grammars are propagating. However, the removal of this restriction causes only at most a linear increase in the size of the grammar, see [11], p. 136.

Here employing synchronized alternating Turing machines we obtain a (single) exponential space upper bound for E0L structural equivalence. However, this result does not yet coincide with the lower bound given by Lemma 4.2.

LEMMA 4.3: *E0L structural equivalence is in $SASPACE\ (n)$.*

Proof: Let $G_i = (V_i, T_i, S_i, P_i)$, $i = 1, 2$, be E0L grammars. We show that a synchronized alternating Turing machine **M** can decide using space linear on $\max\{\text{size}(G_1), \text{size}(G_2)\}$ whether

$$STS_{[p]}(G_1) - STS_{[p]}(G_2) \neq \emptyset. \tag{12}$$

The intuitive idea of the algorithm can be described as follows. The SATM **M** constructs nondeterministically a syntax tree $t_1 \in TS_{[p]}(G_1)$ and simultaneously verifies that for all terminal syntaxt trees t_2 of G_2 we have $\text{str}_{G_1}(t_1) \neq \text{str}_{G_2}(t_2)$. The computation branches universally following the structure of t_1, this makes it possible to use only linear space. Using the synchronization condition, **M** can verify that t_1 corresponds to a parallel derivation.

In the following by an *instantaneous description*, ID, we mean a tuple

$$(A, \{B_1, \ldots, B_m\}),$$

where $A \in V_1$ and $B_1, \ldots, B_m \in V_2$, $0 \leq m \leq \#V_2$. The operation of the STAM **M** is described by the following procedure. The machine stores the contents of the variable Z on its work tape. The machine has two synchronizing symbols $SYNC_1$ and $SYNC_2$.

PROCEDURE. (Operation of **M**.)

BEGIN Set $Z := (S_1, \{S_2\})$.

(I) Assume that $Z = (A, \{B_1, \ldots, B_m\})$, $m \geq 0$. Choose nondeterministically a production

$$A \to A_1 \ldots A_k \in P_1.$$

(II) **(a)** If $A_1 \ldots A_k \in V_1^+$ then produce the synchronizing symbol $SYNC_1$. (That is, **M** goes to an internal state having the synchronizing symbol $SYNC_1$)

(b) If $A_1 \ldots A_k \in T_1^+$ and for all $j \in \{1, \ldots, m\} : (B_j \to A_1 \ldots A_k) \notin P_2$ then produce the synchronizing symbol $SYNC_2$ and halt in an accepting state.

(c) If $k = 0$ and for all $j \in \{1, \ldots, m\} : (B_j \to \lambda) \notin P_2$ then halt in an accepting state, (and do not produce a synchronizing symbol).

(III) (Here we can assume that $k \geq 1$, $A_i \in V_1$, $i = 1, \ldots, k$.) Construct the sets $C_i \subseteq V_2$, $1 \leq i \leq k$, as follows. First set $C_i := \emptyset$, $i = 1, \ldots, k$.

(a) For $j = 1, \ldots, m$ do the following. Let

$$p_r : B_j \to b_1^r \ldots b_k^r, \quad r = 1, \ldots, s, \tag{13}$$

be all the production of P_2 with left-hand side B_j and right-hand side a non-terminal word of length k. For every $r = 1, \ldots, s$, choose $i \in \{1, \ldots, k\}$ and set

$$C_i := C_i \cup \{b_i^r\}.$$

(b) The computation branches universally to k parallel processes. In the ith branch set $Z := (A_i, C_i)$ **and go to (I)**.

END

This completes the description of the machine **M**. Note that above k is not a constant and in (IIIb) the branching into k parallel computations has to be performed in several steps. If in (IIb), (IIc) there exists a production of P_2 of the required type, then the computation halts and rejects.

Claim 2. **M** has an accepting synchronized computation tree T starting from an ID $(A, \{B_1, \ldots, B_m\})$ if and only if

$$STS_{[p]}(G_1[A]) - \bigcup_{i=1}^{m} STS_{[p]}(G_2[B_i]) \neq \emptyset. \tag{14}$$

Proof of Claim 2. We prove the "only if" direction of the claim. Denote by T_1 the tree obtained from the computation tree T by removing the second

components (subset of V_2) from the ID's labeling the nodes and attaching for the leaves of T new successors corresponding to the terminating productions of P_1 that were used in (IIb) and (IIc) to end the computation of the respective branch of T. Every step simulating a non terminating production of P_1 in T_1 forces **M**, according to (IIa), to produce the synchronizing symbol $SYNC_1$ in the computation T. Erasing productions produce no synchronizing symbol and other terminating productions cause **M** to produce the synchronizing symbol $SYNC_2$. Thus the synchronization condition guarantees that all branches of T_1 ending in terminal symbols have the same length l_1 and branches ending with the symbol $\hat{\lambda}$ have length at most l_1. It follows that T_1 corresponds to a parallel derivation, *i.e.*, $T_1 \in TS_{[p]}(G_1[A])$.

Let $A \to A_1 \ldots A_k$ be the production used at the root of T_1, *i.e.*, the production chosen in (I) at the beginning of the computation of T. Denote $\mathrm{str}_{G_1}(T_1) = t$ and let the immediate subtrees of t be $t_1, \ldots, t_k$. We show that

$$t \notin \bigcup_{i=1}^{m} STS_{[p]}(G_2[B_i]). \tag{15}$$

Let $(A_i, \mathcal{C}_i)$, $i = 1, \ldots, k$, be the ID's obtained in the computation tree T after one cycle (I)-(III). Inductively, we assume that Claim 2 holds for $(A_i, \mathcal{C}_i)$, $i = 1, \ldots, k$. This is possible since the base case for single-node computation trees follows from the acceptance conditions of **M** defined in (IIb), (IIc). The sets $\mathcal{C}_i$, $1 \leq i \leq k$, are constructed by (IIIa) and thus for every production $B_j \to b_1 \ldots b_k$, $1 \leq j \leq m$, $(b_a \in V_2, a = 1, \ldots, k)$, there exists $i \in \{1, \ldots, k\}$ such that $b_i \in \mathcal{C}_i$. By the inductive assumption, $t_i \notin STS_{[p]}(G_2[b_i])$. Since the production $B_j \to b_1 \ldots b_k$ is arbitrary, it follows that (15) holds.

Similarly, it can be shown that (14) implies the existence of an accepting synchronized computation starting from the ID $(A, \{B_1, \ldots, B_m\})$. This completes the proof of Claim 2.

By Claim 2, **M** accepts a description of a pair (G_1, G_2) exactly then when (12) holds. We still consider the space requirements of the machine **M**. Let p_1 be the maximal length of the right-hand sides of productions of P_1. The machine **M** needs a maximal amount of space when in (IIIa) it has to store ID's $(A_1, \mathcal{C}_1) \ldots (A_k, \mathcal{C}_k)$ on the worktape, where $k \leq p_1$. From the definition of (IIIa) it follows that $\sum_{i=1}^{k} (\#\mathcal{C}_i)$ is at most the number of productions of P_2. Thus the space needed for the ID's $(A_i, \mathcal{C}_i)$, $i = 1, \ldots, k$, can be linearly bounded by $\max\{\mathrm{size}(G_1), \mathrm{size}(G_2)\}$. $\square$

Combining the previous lemmas and (1) we have:

THEOREM 4.2: *The E0L structural equivalence problem is logspace hard for DEXT and it is in EXPSPACE.*

Also the strong structural equivalence problem for ET0L grammars can be decided by a synchronized alternating Turing machine in linear space. For the definition of the notion of strong structural equivalence we refer the reader to [7]. Let ET0L grammars G_1 and G_2 be given. Similarly as in the proof of Lemma 4.3, a synchronized alternating Turing machine **M** constructs nondeterministically an α-expanded terminal syntax tree of G_1, where α is a sequence of tables of G_1, and simultaneously verifies that, for all α-expanded terminal syntax trees t_2 of G_2, we have $\mathrm{str}_{G_1}(t_1) \neq \mathrm{str}_{G_2}(t_2)$. The only difference from the proof of Lemma 4.3 is that now M needs to verify that the sequence of tables used in different branches of t_1 (and of t_2) is α. This can be done by using different synchronizing symbols for each table of the grammars. (instead of the symbol $SYNC_1$, a computation step of M simulating a nonterminating production of G_1 belonging to a table T_i produces a synchronizing symbol $SYNC_1^{T_i}$.) The machine **M** operates in linear space.

For E0L grammars strong structural equivalence reduces to structural equivalence. Thus by the above observations and Theorem 4.2 we have:

THEOREM 4.3: *The strong structural equivalence problem for ET0L grammars is logspace hard for DEXT and it is in EXPSPACE.*

The structural equivalence problem for E0L grammars is a special case of ET0L strong structural equivalence. Both the upper and the lower bounds obtained above for their complexity are the same. It remains an open question whether E0L structural equivalence and ET0L strong structural equivalence have the same complexity.

REFERENCES

1. J. L. BALCÁZAR, J. DIÁZ and J. GABARRÓ, *Structural Complexity I and II*, EATCS Monographs on Theoretical Computer Science, Vol. 11 and Vol. 22, (Eds. W. Brauer, G. Rozenberg, A. Salomaa), Springer-Verlag, Berlin-Heidelberg, 1988 & 1900.
2. J. DASSOW, J. HROMKOVIC, J. KARHUMAKI, B. ROVAN and A. SLOBODOVÁ, On the power of synchronization in parallel computations, *Proc. of the 14th Symposium on Mathematical Foundations of Computer Science, MFCS'89*, Lect. Notes Comput. Sci., *379*, Springer-Verlag, 1989, pp. 196-206.
3. A. K. CHANDRA, D. C. KOZEN and L. J. STOCKMEYER, Alternation, *J. Assoc. Comput. Math.*, 1981, *28*, pp. 114-133.

4. F. GÉCSEG and M. STEINBY, *Tree automata*, Akadémiai Kiadó, Budapest, 1984.
5. J. HROMKOVIČ, J. KARHUMAKI, B. ROVAN and A. SLOBODOVÁ, On the power of synchronization in parallal computations, *Discrete Appl. Math.*, 1991, *32*, pp. 155-182.
6. J. HROMKOVIČ, B. ROVAN and A. SLOBODOVÁ, Deterministic versus nondeterministic space in terms of synchronized alternating machines, *Theory. Comput. Sci.*, 1994, *132*, pp. 319-336.
7. G. ISTRATE, The strong equivalence of ET0L grammars. In: *Developments in Language Theory*, G. Rozenberg and A. Salomaa (eds.), World, Scientific, 1994, pp. 81-89.
8. N. JONES and S. SKYUM, Complexity of some problems concerning L systems, *Math. Systems Theory*, 1979, *13*, pp. 29-43.
9. K.-J. LANGE and M. SCHUDY, The complexity of the emptiness problem for E0L systems. In: *Lindenmayer systems; Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, G. Rozenberg and A. Salomaa (eds.), Springer, 1992, pp. 167-175.
10. R. MCNAUGHTON, Parenthesis grammars, *J. Assoc. Comput. Mach.*, 1967, *14*, pp. 490-500.
11. V. NIEMI, A normal form for structurally equivalent E0L grammars. In: *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, G. Rozenberg and A. Salomaa (eds.), Springer-Verlag, 1992, pp. 133-148.
12. T. OTTMANN and D. WOOD, Defining families of trees with E0L grammars, *Discrete Applied Math.*, 1991, *32*, pp. 195-209.
13. T. OTTMANN and D. WOOD, Simplifications of E0L grammars. In *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology*, G. Rozenberg and A. Salomaa (eds)., Springer-Verlag, 1992, pp. 149-166.
14. M. PAULL and S. UNGER, Structural equivalence of context-free grammars, *J. Comput. System Sci.*, 1968, *2*, pp. 427-463.
15. G. ROZENBERG and A. SALOMAA, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.
16. K. SALOMAA and S. YU, Decidability of structural equivalence of E0L grammars, *Theoret. Comput. Sci.*, 1991, *82*, pp. 131-139.
17. K. SALOMAA, D. WOOD and S. YU, Structural equivalence and ET0L grammars, *Proc. of the 9th Conference on Fundamentals of Computation Theory*, FCT'93, Lect. Notes Comput. Sci., *710*, Springer-Verlag, 1993, pp. 430-439.
18. H. SEIDL, Deciding equivalence of finite tree automata, *SIAM J. Comput.*, 1990, *19*, pp. 424-437.
19. A. SLOBODOVÁ, Communication for alternating machines, *Acta Inform.*, 1992, *29*, pp. 425-441.
20. J. W. TATCHER, Tree automata: an informal survey. In: *Currents in the Theory of Computing*, A. V. Aho (ed.), Prentice Hall, Englewood Cliffs, NJ, 1973, pp. 143-172.
21. J. VAN LEEUWEN, The membership question for ET0L languages is polynomially complete, *Inform. Process. Lett.*, 1975, *3*, pp. 138-143.
22. J. VAN LEEUWEN, The tape-complexity of context-independent developmental languages, *J. Comput. System. Sci.*, 1975, *11*, pp. 203-211.
23. J. WIEDERMANN, On the power of synchronization, *J. Inf. Process. Cybern. EIK*, 1989, *25*, pp. 499-506.
24. D. WOOD, *Theory of Computation*, John Wiley & Sons, New York, 1987. (Second edition, 1996, in preparation.)