INFORMATIQUE THÉORIQUE ET APPLICATIONS

KAI SALOMAA

# Yield-languages recognized by alternating tree recognizers

*Informatique théorique et applications*, tome 22, nº 3 (1988), p. 319-339

<http://www.numdam.org/item?id=ITA_1988__22_3_319_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 22, n° 3, 1988, p. 319 à 339)

# YIELD-LANGUAGES RECOGNIZED BY ALTERNATING TREE RECOGNIZERS (*)

by Kai SALOMAA [1]

Communicated by J. BERSTEL

Abstract. – *We show that every context-sensitive language is the yield of a tree language recognized by an alternating bottom-up tree recognizer. If one nullary symbol of the ranked alphabet is defined to correspond to the empty word in the yield, then every recursively enumerable language can be represented as the yield of a tree language recognized by an alternating recognizer.In establishing these results we use the observation that in an alternating bottom-up computation the tree recognizer is able to check that every configuration appearing in the computation tree belongs to a given regular tree language.*

Résumé. – *Nous montrons que tout langage context-sensitif est langage des feuilles d'un langage d'arbre reconnu par un automate d'arbres alternant ascendant. Si un symbole zéroaire a pour image le mot vide, alors tout langage récursivement énumérable peut être représenté comme le langage des feuilles d'un langage d'arbre reconnu par un automate alternant. Dans les preuves, on utilise le fait qu'un automate d'arbre est capable de tester si les configurations produites dans un calcul alternant ascendant appartiennent à un langage d'arbre régulier donné.*

## INTRODUCTION

Alternation has been studied as a generalization of nondeterminism for many machine models. In [2,5] it is shown that alternation does not increase the power of nondeterministic one-way or two-way finite automata. In the same way it is proved in [11] that alternating top-down tree automata just recognize the regular forests. (We call tree languages forests for short.) Alternating automata operating on infinite trees are studied in [6].

Alternating bottom-up tree recognizers are discussed in [10] where it is shown that the family of recognized forests properly contains the regular

(*) Received in July 1986, revised in October 1987.
This research was supported by the Academy of Finland and the Heikki and Hilma Honkanen Foundation.
[1] University of Turku, Department of Mathematics, SF-20500 Turku, Finland.

forests. The intuitive reason for this is that in an alternating bottom-up computation the order in which different subtrees of the input tree are read can be essential; therefore an alternating bottom-up computation cannot be simulated by a deterministic computation using a standard subset construction.

Here we study the yield-languages of forests recognized by alternating bottom-up recognizers. In the following we call these languages shortly alternating yield-languages. The yield-languages of regular and algebraic forests respectively are the context-free and indexed languages, *cf.* [3,8]. The family of forests recognized by alternating bottom-up recognizers is incomparable with the family of algebraic forests, *cf.* [10], but in section 4 we will show that every context-sensitive language can be represented as an alternating yield-language, and hence the family of alternating yield-languages properly contains the indexed languages. The question whether all alternating yield-languages are context-sensitive is open, but we have the partial result that given an alternating yield-language one cannot effectively construct a context-sensitive grammar for it.

In section 5 we show that if nodes of the trees may be labeled also by the empty word, then every recursively enumerable language is the yield of a forest recognized by an alternating bottom-up recognizer. In proving these results we use a normal form for context-sensitive grammars obtained in [7].

In the first section we establish some notation. In section 2 we recall the definition of an alternating bottom-up recognizer from [10] and define the alternating yield-languages. In the third section it is shown that regular control does not increase the power of alternating bottom-up computation. This result will considerably simplify the proofs in sections 4 and 5.

## 1. PRELIMINARIES

The reader is assumed to be familiar with some basic notions concerning formal languages, trees and forests, *cf.* [3, 4, 9]. Here we just fix the notation and briefly recall some definitions.

Let $A$ be a set. Then $\mathscr{P}(A)$ denotes the set of subsets of $A$ and $\bar{A}$ is defined to be the set $\{\bar{a} \mid a \in A\}$. We may assume that always $A \cap \bar{A} = \varnothing$. If $\omega \subseteq A \times A$ is a binary relation, then $\omega^*$ denotes the reflexive, transitive closure of $\omega$. The symbol $\lambda$ denotes the empty word. Symbols $\Sigma$ and $\Omega$ always denote finite ranked alphabets. The set of $m$-ary ($m \geqq 0$) symbols of $\Sigma$ is denoted by

$\Sigma_m$. The set of $\Sigma$-*trees* $F_\Sigma$ is the smallest set $S$ such that:

(i) $\Sigma_0 \subseteq S$, and

(ii) if $m \geqq 1$, $\sigma \in \Sigma_m$, and $t_1, \ldots, t_m \in S$, then $\sigma(t_1, \ldots, t_m) \in S$.

A $\Sigma$-*forest* is a subset of $F_\Sigma$. The family of regular forests is denoted by *REG*. The set of $\Sigma A$-trees $F_\Sigma(A)$ is defined to be $F_\Omega$, where $\Omega_0 = \Sigma_0 \cup A$ and $\Omega_m = \Sigma_m$ when $m \geqq 1$.

The *yield-function*

$$yd_\Sigma: \quad F_\Sigma \to \Sigma_0^+$$

is defined as follows:

(i) $yd_\Sigma(b) = b$ when $b \in \Sigma_0$.

(ii) If $m \geqq 1$, $\sigma \in \Sigma_m$ and $t_1, \ldots, t_m \in F_\Sigma$, then

$$yd_\Sigma(\sigma(t_1, \ldots, t_m)) = yd_\Sigma(t_1) \ldots yd_\Sigma(t_m).$$

We simply denote $yd_\Sigma$ by $yd$ when $\Sigma$ is clear from the context.

For every $c \in \Sigma_0$ we define the *c-yield-function*

$$yd_c: \quad F_\Sigma \to (\Sigma_0 - \{c\})^*$$

as follows:

(i) $yd_c(c) = \lambda$.

(ii) $yd_c(b) = b$ if $b \in \Sigma_0 - \{c\}$.

(iii) Let $t = \sigma(t_1, \ldots, t_m)$, where $m \geqq 1$, $\sigma \in \Sigma_m$ and $t_1, \ldots, t_m \in F_\Sigma$. Then

$$yd_c(t) = yd_c(t_1) \ldots yd_c(t_m).$$

The word $yd(t)$ is obtained from $t$ simply by concatenating the labels of the leaves of $t$ from the left to the right. In the word $yd_c(t)$ all occurrences of $c$ are omitted.

## 2. ALTERNATING BOTTOM-UP TREE RECOGNIZERS

First we recall the definition of an alternating bottom-up recognizer, *cf.* [10].

DEFINITION 2.1: An *alternating bottom-up tree recognizer* is a four-tuple $\underline{A} = (A, \Sigma, g, A')$, where

(i) $A$ is a finite set of states.

(ii) $\Sigma$ is a finite ranked alphabet.

(iii) $g$ is a function that associates with every element $\sigma \in \Sigma_m (m \geqq 0)$ a mapping $\sigma_g : A^m \to \mathcal{P}(\mathcal{P}(A))$. (If $\sigma \in \Sigma_0$, $\sigma_g$ is interpreted to be an element of $\mathcal{P}(\mathcal{P}(A))$.)

(iv) $A' \subseteq A$ is the set of accepting final states.

The family of alternating bottom-up recognizers is denoted by $AR$.

In the following definitions $\underline{A} = (A, \Sigma, g, A')$ is as above.

DEFINITION 2.2: $\Sigma A$-trees are called *$\underline{A}$-configurations*. The set of *active subtrees* of an $\underline{A}$-configuration K, *act*(*K*), consists of all occurrences of subtrees of $K$ of the form $\sigma(a_1, \ldots, a_m)$, $m \geqq 0$, $\sigma \in \Sigma_m$, $a_1, \ldots, a_m \in A$. When $f = \sigma(a_1, \ldots, a_m) \in \text{act}(K)$, we write

$$f_g = \sigma_g(a_1, \ldots, a_m) \, (\in \mathcal{P}(\mathcal{P}(A))).$$

A *configuration tree* of $\underline{A}$ is a finite rooted tree (in the sense of graph theory), the nodes being labeled by $\underline{A}$-configurations. The set of configuration trees of $\underline{A}$ is denoted by $CT(\underline{A})$. If $T \in CT(\underline{A})$, *conf*(*T*) denotes the set of all $\underline{A}$-configurations that occur as a label of some node of $T$.

For a recognizer $\underline{A}$ we define the $EU$- and $UE$-modes of operation. In the $EU$-mode at each computation step $\underline{A}$ first makes an existential choice and thereafter branches universally; in the $UE$-mode $\underline{A}$ first branches universally and then makes the existential choices.

DEFINITION 2.3: The transition relations of the recognizer $\underline{A}$, $\vdash_{\underline{A}}^{EU}$ and $\vdash_{\underline{A}}^{UE} \subseteq CT(\underline{A}) \times CT(\underline{A})$, are defined as follows. Let $Z \in \{EU, UE\}$ and $T$, $T' \in CT(\underline{A})$. Then

$$T \vdash_{\underline{A}}^{Z} T' \text{ iff } T' \text{ is obtained from } T \text{ as follows:}$$

Choose a leaf $n$ of $T$ and suppose that $n$ is labeled by a configuration $K \in F_\Sigma(A)$. Choose an active subtree $f = \sigma(a_1, \ldots, a_m)$ of $K$, $(m \geqq 0)$.

(i) Case $Z = EU$: Choose a nonempty set $\{c_1, \ldots, c_h\} \in f_g$, $h > 0$. Now $T'$ is obtained from $T$ by attaching for the node $n$ $h$ successors that are respectively labeled by the configurations $K(f \leftarrow c_1), \ldots, K(f \leftarrow c_h)$. [Here $K(f \leftarrow c_i)$ denotes the $\Sigma A$-tree that is obtained from $K$ by replacing the fixed occurrence of $f$ with $c_i$.]

(ii) Case $Z = UE$: Let $f_g = \{D_1, \ldots, D_p\}$, $D_i \subseteq A$, $i = 1, \ldots, p$. Choose elements $d_i \in D_i$, $i = 1, \ldots, p$. Now in $T'$ the node $n$ has $p$ successors that are labeled by the configurations $K(f \leftarrow d_1), \ldots, K(f \leftarrow d_p)$.

DEFINITION 2.4: *Let* $K \in F_\Sigma(A)$ and $Z \in \{EU, UE\}$. The set of *Z-alternating K-computation trees of* $\underline{A}$ is

$$\mathrm{COM}_Z(\underline{A}, K) = \{ T \in CT(\underline{A}) \mid K (\vdash_{\underline{A}}^{Z})^* T \}.$$

(Here $K$ is interpreted to be the configuration tree consisting of only one node labeled by $K$.)

A computation tree is said to be *accepting* if all its leaves are labeled by elements of $A'$. The configuration $K$ is *Z-accepting* if $\mathrm{COM}_Z(\underline{A}, K)$ contains at least one accepting computation tree. The set of $Z$-accepting configurations is denoted by $H_Z(\underline{A})$. The forest *Z-recognized* by $\underline{A}$ is

$$L_Z(\underline{A}) = H_Z(\underline{A}) \cap F_\Sigma.$$

The corresponding family of forests is denoted

$$\mathscr{L}_Z(AR) = \{ L_Z(\underline{A}) \mid \underline{A} \in AR \}.$$

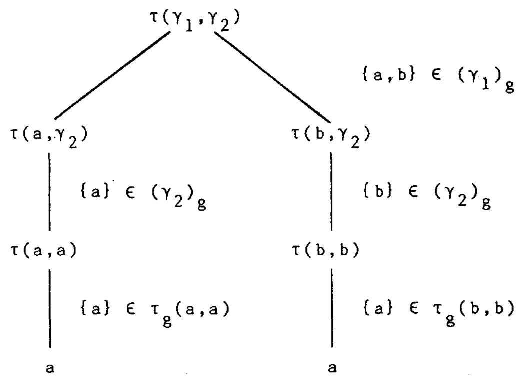
*Example* 2.5: Let $\underline{A} = (A, \Sigma, g, A')$, where $A = \{a, b\}$, $A' = \{a\}$, $\Sigma = \Sigma_2 \cup \Sigma_0$, $\Sigma_2 = \{\tau\}$, $\Sigma_0 = \{\gamma_1, \gamma_2\}$ and $g$ is defined as follows:

$(\gamma_1)_g = \{\{a, b\}\}$;

$(\gamma_2)_g = \{\{a\}, \{b\}\}$;

$\tau_g(a, a) = \tau_g(b, b) = \{\{a\}\}$;

$\tau_g(a, b) = \tau_g(b, a) = \varnothing$.




Let $t = \tau(\gamma_1, \gamma_2)$. Then Figure represents an $EU$-accepting $t$-computation tree of $\underline{A}$.

In [10] it is shown that $\mathscr{L}_{UE}(AR) \subseteq \mathscr{L}_{EU}(AR)$ and that corresponding to every $L \in \mathscr{L}_{EU}(AR)$ there exists a forest $L' \in \mathscr{L}_{UE}(AR)$ such that $yd(L') = yd(L)$. Hence we have

THEOREM 2.6: $yd(\mathscr{L}_{EU}(AR)) = yd(\mathscr{L}_{UE}(AR))$.

$$(Here\ yd(\mathscr{L}) = \{ yd(L) \mid L \in \mathscr{L} \}.)$$

The languages of $yd(\mathscr{L}_Z(AR))$ are called *Z-alternating yield-languages*, $Z \in \{ EU, UE \}$. By Theorem 2.6 the families of *EU*- and *UE*-alternating yield-languages are equal and we just call them *alternating yield-languages.*

Let $Z \in \{ EU, UE \}$. We say that $L$ is a *Z-alternating extended yield-language* if there exists a recognizer $\underline{A} = (A, \Sigma, g, A')$ and $c \in \Sigma_0$ such that

$$L = yd_c(L_Z(\underline{A})).$$

If $M_1, M_2 \subseteq F_\Sigma$, $yd(M_1) = yd(M_2)$ and $c \in \Sigma_0$, then $yd_c(M_1) = yd_c(M_2)$. Hence from Theorem 2.6 it follows also that the families of *EU*- and *UE*-alternating extended yield-languages are equal. They are called in the following *alternating extended yield-languages.*

Finally we present some definitions needed in the following sections.

DEFINITION 2.7: Let $Z \in \{ EU, UE \}$ and $\underline{A} = (A, \Sigma, g, A')$, $\underline{B} = (B, \Sigma, h, B')$. Suppose that $A \subseteq B$ and $K \in F_\Sigma(A)$. Let $T_1 \in \mathrm{COM}_Z(\underline{A}, K)$ and $T_2 \in \mathrm{COM}_Z(\underline{B}, K)$ (note that $K$ is also a $\underline{B}$-configuration). We say that $T_1$ *thins* $T_2$ if

(i) $T_1$ is obtained from $T_2$ by deleting some nodes (possibly an empty set) together with all their successors, and

(ii) every leaf of $T_1$ is also a leaf in $T_2$.

(Intuitively this means that $T_1$ is obtained from $T_2$ by "cutting off some subtrees" in such a way that if a non-leaf node $n$ of $T_2$ is not cut off, then also some immediate successor of $n$ is not cut off.)

DEFINITION 2.8: Let $\underline{A} = (A, \Sigma, g, A') \in AR$. The recognizer $\underline{A}$ is said to be *complete* if for all $m \geqq 0$, $\sigma \in \Sigma_m$, and $a_1, \ldots, a_m \in A$:

$$\sigma_g(a_1, \ldots, a_m) \neq \varnothing \text{ and } \varnothing \text{ does not belong to } \sigma_g(a_1, \ldots, a_m).$$

The recognizer $\underline{A}$ is *deterministic* if for all $m \geqq 0$, $\sigma \in \Sigma_m$, $a_1, \ldots, a_m \in A$:

$$\sigma_g(a_1, \ldots, a_m) = \{\{ a \}\},$$

where $a \in A$.

Clearly the above definition is equivalent to the usual definition of a deterministic bottom-up tree recognizer, *cf.* [3]. Also nondeterministic tree recognizers can be obtained as a subcase of alternating recognizers. An arbitrary nondeterministic recognizer can be simulated by the $EU$-computation (resp. $UE$-computation) of an $AR$-recognizer $\underline{A}=(A, \Sigma, g, A')$, where for every $\sigma \in \Sigma_m$ and $a_1, \ldots, a_m \in A$, $\sigma_g(a_1, \ldots, a_m)$ consists of one-element subsets of $A$ (resp. consists of one subset of $A$). The deterministic as well as nondeterministic bottom-up tree recognizers exactly recognize the family of regular forests. On the other hand, alternating recognizers generally can recognize also nonregular and even nonalgebraic forests, *cf.* [10].

## 3. CONTROL-FORESTS FOR ALTERNATING COMPUTATION

Here we show that an $AR$-recognizer $\underline{A}$ is able to check that every configuration occurring in accepting computation trees of $\underline{A}$ belongs to a given regular forest. For instance nondeterministic tree recognizers do not have this property. Regular control-forests will be very useful in constructing alternating recognizers for nonregular forests.

DEFINITION 3.1: Let $\underline{A}=(A, \Sigma, g, A') \in AR$, $Z \in \{EU, UE\}$ and $K \in F_\Sigma(A)$. Suppose that $M \subseteq F_\Sigma(A)$. The set of *M-controlled* Z-alternating $K$-computation trees of $\underline{A}$ is

$$\mathrm{COM}_Z(\underline{A}, K)[M]=\{T \in \mathrm{COM}_Z(\underline{A}, K) \mid \mathrm{conf}(T) \subseteq M\}.$$

The configuration $K$ is *M-controlled Z-accepting* if $\mathrm{COM}_Z(\underline{A}, K)[M]$ contains an accepting computation tree. The set of $M$-controlled $Z$-accepting configurations is denoted by $H_Z(\underline{A})[M]$ and the forest *M-controlled Z-recognized* by $\underline{A}$ is

$$L_Z(\underline{A})[M]=H_Z(\underline{A})[M] \cap F_\Sigma.$$

The family of forests $Z$-recognized by an $AR$-recognizer with a regular control-forest is denoted by $\mathscr{L}_Z(AR)[\mathrm{REG}]$.

LEMMA 3.2: *Let $Z \in \{EU, UE\}$. For every $\underline{A}=(A, \Sigma, g, A') \in AR$ and $M \subseteq F_\Sigma(A)$ there exist a complete recognizer $\underline{B}=(B, \Sigma, h, B')$ and $M_1 \subseteq F_\Sigma(B)$ such that*

$$L_Z(\underline{A})[M]=L_Z(\underline{B})[M_1].$$

*In addition $M_1$ can be chosen to be regular if $M$ is regular.*

*Proof.* — We omit the proof which is quite straightforward. (One can choose $B=A\cup\{d\}$, where $d$ is a "dead state", and $M_1=M$.) □

We show that $\mathscr{L}_{EU}(AR)$ is closed with respect to regular control. The intuitive idea of the proof is that in a computation step in a configuration $K$ an $AR$-recognizer can make one additional universal branching to the computation tree, and this branch can be made to check that $K$ belongs to the given regular forest.

THEOREM 3.3: $\mathscr{L}_{EU}(AR)[\text{REG}]=\mathscr{L}_{EU}(AR)$.

*Proof.* — Clearly $\mathscr{L}_{EU}(AR)\subseteq\mathscr{L}_{EU}(AR)[\text{REG}]$. We prove the converse inclusion. Let $L\in\mathscr{L}_{EU}(AR)[\text{REG}]$. By Lemma 3.2 there exist a complete recognizer $\underline{A}=(A,\Sigma,g,A')$ and a regular forest $M\subseteq F_\Sigma(A)$ such that $L=L_{EU}(\underline{A})[M]$.

Let $\underline{B}=(B,\Omega,d,B')$ be a deterministic recognizer such that $L_{EU}(\underline{B})=M$. Here $\Omega_m=\Sigma_m$ if $m>0$ and $\Omega_0=\Sigma_0\cup A$. Without restriction we may assume that $B\cap A=\emptyset$.

Let $m\geqq 0$, $\sigma\in\Omega_m$, $b_1,\ldots,b_m\in B$ and suppose that

$$\sigma_d(b_1,\ldots,b_m)=\{\{b\}\},b\in B.$$

We make the convention that in the following $\sigma_d(b_1,\ldots,b_m)$ denotes always just the element $b$ (and not $\{\{b\}\}\in\mathscr{P}(\mathscr{P}(B))$).

We construct a recognizer $\underline{C}=(C,\Sigma,h,C')$ such that $L_{EU}(\underline{C})=L_{EU}(\underline{A})[M]$. We choose

$$C=A\cup B\cup\bar{B}\qquad\text{and}\qquad C'=(A'\cap M)\cup\bar{B}'.$$

The function $h$ is defined in (1)-(4).

Let $m\geqq 0$, $\sigma\in\Sigma_m$, $a_1,\ldots,a_m\in A$ and let $f=\sigma(a_1,\ldots,a_m)$. We denote $f_d=\sigma_d((a_1)_d,\ldots,(a_m)_d)$ $(\in B)$. Then

$$(1)\qquad f_h=\{D\cup\{\bar{f}_d\}\mid D\in f_g\}\cup\{\{f_d\}\}.$$

For every element $c\in C$ we define $\tilde{c}\in B$ as follows:

$$\tilde{c}=\begin{cases}c_d & \text{if } c\in A\,(\subseteq\Omega_0).\\ c & \text{if } c\in B.\\ c_1 & \text{if } c=\bar{c}_1\in\bar{B}.\end{cases}$$

Now let $m\geqq 1$, $\sigma\in\Sigma_m$, $c_1,\ldots,c_m\in C$ and suppose that at least one of the elements $c_i$, $1\leqq i\leqq m$, does not belong to $A$. We have the following three possibilities.

(2) If $c_1, \ldots, c_m \in A \cup B$, then

$$\sigma_h(c_1, \ldots, c_m) = \{\{\sigma_d(\tilde{c}_1, \ldots, \tilde{c}_m)\}\}.$$

(3) If there exists an $i$, $1 \leqq i \leqq m$, such that $c_i \in \bar{B}$ and $c_j \in A \cup B$ when $j \neq i$, then

$$\sigma_h(c_1, \ldots, c_m) = \{\{\bar{b}\}\},$$

where $b = \sigma_d(\tilde{c}_1, \ldots, \tilde{c}_m)$.

(4) If there exist $i, j \in \{1, \ldots, m\}$, $i \neq j$ such that $c_i$, $c_j \in \bar{B}$, then

$$\sigma_h(c_1, \ldots, c_m) = \varnothing.$$

The intuitive idea of the construction is roughly as follows. As it reads an active subtree $f$ of an arbitrary $\underline{A}$-configuration $K$, $\underline{C}$ simulates a computation step of $\underline{A}$ by an existential choice of the form $D \cup \{\bar{f}_d\}$, $D \in f_g$. The computation from nodes $K(f \leftarrow d)$, $d \in D$, continues simulating the computation of $\underline{A}$, and the computation starting from $K(f \leftarrow \bar{f}_d)$ checks that $K \in L_{EU}(\underline{B}) = M$.

**Fact 1**

Let $K \in F_\Sigma(C)$ be such that

$$yd(K) \in (\Sigma_0 \cup A \cup B)^* B (\Sigma_0 \cup A \cup B)^*,$$

(i. e., at least one leaf of $K$ is labeled by an element of $B$, and none is labeled by an element of $\bar{B}$.) Then $K$ is not in $H_{EU}(\underline{C})$.

*Proof of Fact* 1: From the rules (1) and (2) it follows that in every $K$-computation tree there exists a branch all the nodes of which are labeled by configurations $K'$ such that $yd(K') \in (\Sigma_0 \cup A \cup B)^* B (\Sigma_0 \cup A \cup B)^*$. [Note that always in (1) $D \neq \varnothing$ since $\underline{A}$ is complete.] Hence this computation path cannot end in an accepting final state.

**Fact 2**

Suppose that $K \in F_\Sigma(A)$ and $\underline{C}$ reads an active subtree $f$ of $K$ making the choice $\{f_d\}$ by (1). Then the resulting configuration $K(f \leftarrow f_d)$ is not in $H_{EU}(\underline{C})$.

*Proof of Fact* 2: This follows immediately from Fact 1 since $f_d$ belongs to $B$.

**Fact 3**

Let $t \in F_\Sigma$ and $T \in \mathrm{COM}_{EU}(\underline{C}, t)$. Suppose that $\mathrm{conf}(T) \subseteq H_{EU}(\underline{C})$. Then there exists $T' \in \mathrm{COM}_{EU}(\underline{A}, t)$ such that $T'$ thins $T$.

*Proof of Fact* 3: Suppose that $t (\vdash_{\underline{C}}^{EU})^n T$, $n \geqq 0$. We prove Fact 3 using induction on $n$.

(i) If $n=0$, then $T$ consists of one node labeled by $t$ and we can choose $T'=T$.

(ii) Suppose that $n=k+1$ and that the claim holds for $n \leqq k$. Let $T_1$ be such that

$$t (\vdash_{\underline{C}}^{EU})^k T_1 \vdash_{\underline{C}}^{EU} T.$$

By the induction assumption there exists $T'_1 \in \mathrm{COM}_{EU}(\underline{A}, t)$ such that $T'_1$ thins $T_1$. Suppose that $T$ is obtained from $T_1$ by adjoining successors to a leaf $n$ of $T_1$.

(i′) If $n$ does not appear in $T'_1$, then $T'_1$ thins $T$.

(ii′) Suppose that $n$ appears in $T'_1$. Then $n$ is labeled by an $\underline{A}$-configuration $K$ and, by Fact 2, when the recognizer $\underline{C}$ in the computation tree $T$ reads an active subtree $f$ of $K$, it makes a choice of the form $D \cup \{\bar{f}_d\}$, $D \in f_g$. Let $T' \in \mathrm{COM}_{EU}(\underline{A}, t)$ be the computation tree that is obtained from $T'_1$ by adjoining the successors $K(f \leftarrow x)$, $x \in D$, for the leaf $n$. Then $T'$ thins $T$.

**Fact 4**

$L_{EU}(\underline{C}) \subseteq L_{EU}(\underline{A})[M]$.

*Proof of Fact* 4: Let $t \in F_\Sigma$ and suppose that $T \in \mathrm{COM}_{EU}(\underline{C}, t)$ is accepting. By Fact 3 there exists $T' \in \mathrm{COM}_{EU}(\underline{A}, t)$ such that $T'$ thins $T$. Now all the leaves of $T'$ must be labeled by elements of $A' \cap M$, and hence $T'$ is an accepting computation tree of $\underline{A}$ and $t \in L_{EU}(\underline{A})$.

Let $K \in \mathrm{conf}(T')$. We claim that $K \in M$.

(i) If $K$ labels a leaf of $T'$, then $K \in A' \cap M$.

(ii) Suppose that $K$ labels a node of $T'$ that is not a leaf. Suppose that when continuing the computation of $T$, $\underline{C}$ reads an active subtree $f$ of $K$. By Fact 2, $\underline{C}$ has to make an existential choice of the form $D \cup \{\bar{f}_d\}$, $D \in f_g$. We consider the subtree of $T$ with the root labeled by $K(f \leftarrow \bar{f}_d)$. Clearly in this subtree whenever the rule (1) is applied, $\underline{C}$ has to make a choice of the form

$\{f'_d\}$. [Otherwise in some branch the computation would lead to a configuration $K'$ containing two nodes labeled by elements of $\bar{B}$, and by (3) and (4) $K'$ would not be accepting.] Hence the computation starting from $K(f \leftarrow \bar{f}_d)$ is performed "as by $\underline{B}$", and it checks that $K \in M$.

**Fact 5**

$H_{EU}(\underline{A})[M] \subseteq H_{EU}(\underline{C})$.

*Proof of Fact* 5: Let $K \in F_\Sigma(A)$ and $T$ be an $EU$-accepting $M$-controlled $K$-computation tree of $\underline{A}$. We show that $K \in H_{EU}(\underline{C})$ using induction on ${}^\# T$ (= the number of nodes of $T$):

(i) If ${}^\# T = 1$, then $K \in A' \cap M \subseteq H_{EU}(\underline{C})$.

(ii) Suppose that ${}^\# T = k+1$, $k \geq 1$. Suppose that at the root of $T$ $\underline{A}$ reads an active subtree $f$ of $K$ making a choice $D \in f_g$. We construct an $EU$-accepting $K$-computation tree $T_1$ of $\underline{C}$ as follows. At the root of $T_1$ $\underline{C}$ reads the active subtree $f$ of $K$ by rule (1) making the choice $D \cup \{\bar{f}_d\}$. By the silent induction assumption the configurations $K(f \leftarrow x)$, $x \in D$, belong to $H_{EU}(\underline{C})$. As it continues the computation of $T_1$ from $K(f \leftarrow \bar{f}_d)$, always when using a rule (1) $\underline{C}$ makes an existential choice of the form $\{f'_d\}$ and otherwise uses the deterministic rules (2) and (3). Since $K \in M$, this computation ends at the root of $K$ in a state $\bar{b}$, $b \in B'$. Hence $K \in H_{EU}(\underline{C})$.

Now from Facts 4 and 5 it follows that

$$L_{EU}(\underline{A})[M] = L_{EU}(\underline{C}). \quad \square$$

Similarly as in Theorem 3.3 it can be shown that also $\mathscr{L}_{UE}(AR)$ is closed with respect to regular control. Here we do not need this result since by Theorem 2.6 the families of $EU$- and $UE$-alternating yield-languages are equal.

## 4. REPRESENTATION OF CONTEXT-SENSITIVE LANGUAGES

In this section we show that every context-sensitive (i. e., type 1, *cf.* [4, 9]) language is an alternating yield-language. The question whether the inclusion holds in the other direction is open, but we have the weaker result that at least there does not exist an algorithm that given an alternating yield-language $L$ (i. e., a recognizer of the corresponding forest) would produce a context-sensitive grammar generating $L$.

Definition 4.1: *A Penttonen normal form context-sensitive grammar* or *PNF grammar* is a four-tuple

$$G=(V, N, S, P),$$

where:

(i) $V$ is a finite alphabet of terminal symbols;

(ii) $N$ is a finite nonterminal alphabet;

(iii) $S \in N$ is the initial nonterminal;

(iv) $P$ is a finite set of productions of the forms

$$B \rightarrow CD, \qquad DB \rightarrow DC \quad \text{or} \quad B \rightarrow b,$$

where $B$, $C$, $D \in N$ and $b \in V$. In addition $P$ may contain the production $S \rightarrow \lambda$ in which case $S$ does not appear in the right-hand side of any production of $P$.

The productions of $P$ define in the usual way the rewrite-relation $\Rightarrow_G \subseteq (V \cup N)^* \times (V \cup N)^*$, *cf.* [4, 9], and the language generated by $G$ is

$$L(G)=\{ w \in V^* \mid S \Rightarrow_G^* w \}.$$

In [7] it is shown that every context-sensitive language not containing the empty word can be generated by a PNF grammar without the rule $S \rightarrow \lambda$. Hence every context-sensitive language $L$ can be generated by a PNF grammar $G$ as in Definition 4.1, and $S \rightarrow \lambda \in P$ iff $\lambda \in L$.

In the following we define the set of derivation trees $D(G)$ of a PNF grammar $G$ and construct an AR-recognizer $\underline{A}$ recognizing the derivation trees of terminal words. We could define some kind of an "empty tree" $t_0$ to correspond to the derivation $S \rightarrow \lambda$, and the recognizer $\underline{A}$ could easily check whether $t_0 \in D(G)$. To simplify the notations, in the rest of this section we always assume that a context-sensitive language does not contain the empty word and that a PNF grammar is without the production $S \rightarrow \lambda$. By what was said above, this is not an essential restriction.

It should be noted that the tree grammar given in Definition 4.2 is not a very natural one. This is because there is no natural way to present context-sensitive derivations as trees.

Definition 4.2: Let $G=(V, N, S, P)$ be a PNF grammar. We define a ranked alphabet $\Omega=\Omega_2 \cup \Omega_1 \cup \Omega_0$ associated with $G$ as follows:

$$\Omega_0=V \cup N, \qquad \Omega_1=\{\sigma\} \quad \text{and} \quad \Omega_2=\{\tau\},$$

where $\sigma$ and $\tau$ are new symbols.

We define a production set $\bar{P}$ that consists of so called unlabeled productions

$$B \to t \quad (B \in N, t \in F_\Omega)$$

and labeled productions of the form

$$B \to^D t \quad (B, D \in N, t \in F_\Omega).$$

The set $\bar{P}$ exactly consists of all productions defined in (1)-(3):

(1) For every production $B \to b \in P$ there is a production $B \to \sigma(b)$ in $\bar{P}$.

(2) For every $B \to CD \in P$ there is $B \to \sigma(\tau(C, D))$ in $\bar{P}$.

(3) For every production $DB \to DC$ of $P$ there is the labeled production $B \to^D \sigma(\sigma(C))$ in $\bar{P}$.

The relation $\vdash_G \subseteq F_\Omega \times F_\Omega$ is defined as follows. Let $t_1, t_2 \in F_\Omega$.

Then $t_1 \vdash_G t_2$ iff $t_2$ is obtained from $t_1$ in one of the following ways:

(i) Replace some occurrence of an element $B \in N$ in $t_1$ by $t$, where $B \to t \in \bar{P}$.

(ii) Replace some occurrence of $B \in N$ in $t_1$ by $t$, where $B \to^D t \in \bar{P}$ and the leaf of $t_1$ immediately to the left of $B$ is labeled by $D$.

Now the set of *derivation trees* of $G$ is defined to be

$$D(G) = \{ t \in F_\Omega \mid S \vdash_G^* t \}.$$

The derivation trees correspond in an obvious way to derivations of $G$ and we have immediately

LEMMA 4.3: *Let $G = (V, N, S, P)$ be a* PNF *grammar and $w \in (V \cup N)^+$. Then $S \Rightarrow_G^* w$ iff there exists a $t \in D(G)$ such that $yd(t) = w$.*

Note that the right-hand sides of productions of $\bar{P}$ would seem to contain "unnecessary" symbols $\sigma$. These are included just to simplify the proof of the following lemma.

LEMMA 4.4: *Let $G = (V, N, S, P)$ be a* PNF *grammar. Then there exists a ranked alphabet $\Sigma$ and a $\Sigma$-forest $M$ such that $M \in \mathscr{L}_{EU}(AR)$ and $yd(M) = L(G)$.*

*Proof.* — Let $\sigma$, $\tau$ and $\Omega$ be as in Definition 4.2. We choose $\Sigma=\Sigma_2 \cup \Sigma_1 \cup \Sigma_0$, where $\Sigma_2=\{\tau\}$, $\Sigma_1=\{\sigma\}$ and $\Sigma_0=V$. Define

$$\underline{A}=(A, \Sigma, g, A'),$$

where

$$A=N \cup \bar{N} \cup (N \times N),$$
$$A'=\{S\}$$

and $g$ is defined as follows.

Let $b \in \Sigma_0$. Then

$$b_g=\{\{\bar{B}\} \mid B \rightarrow b \in P\}. \tag{4}$$

Let $B$, $C \in N$. Then

$$\sigma_g(B)=\{\{(E, B)\} \mid (\exists D \in N): DE \rightarrow DB \in P, E \in N\} \tag{5}$$

$$\sigma_g(\bar{B})=\{\{B\}\} \tag{6}$$

$$\sigma_g((B, C))=\{\{B\}\} \tag{7}$$

$$\tau_g(B, C)=\{\{\bar{D}\} \mid D \rightarrow BC \in P, D \in N\} \tag{8}$$

$$\tau_g(x, y)=\varnothing \text{ if } x \text{ or } y \text{ belongs to } \bar{N} \cup (N \times N). \tag{9}$$

We choose

$$M_1=(V \cup N)^* \cup (V \cup N)^* \bar{N} (V \cup N)^* \cup (V \cup N)^* \{D (B, C) \mid DB \rightarrow DC \in P\} (V \cup N)^*$$

and define $M \subseteq F_\Sigma(A)$ by

$$M=yd^{-1}(M_1). \tag{10}$$

Since $M_1$ is a regular language, we have $M \in \mathrm{REG}$. Now we claim that

$$H_{EU}(\underline{A})[M] \cap F_\Sigma(N)=D(G). \tag{11}$$

Let $t \in D(G)$. Then $t \in F_\Sigma(N)$ since $D(G) \subseteq F_\Omega=F_\Sigma(N)$. Suppose that $S(\vdash_G)^n t$, $n \geqq 0$. Using induction on $n$ we show that $t \in H_{EU}(\underline{A})[M]$.

If $n=0$, then $t=S \in A' \cap M \subseteq H_{EU}(\underline{A})[M]$. Next suppose that $n=k+1$ and that $t_1 \in H_{EU}(\underline{A})[M]$ whenever $S(\vdash_G)^k t_1$. Let $t'$ be such that

$$S(\vdash_G)^k t' \vdash_G t.$$

Suppose that $t$ is obtained from $t'$ by replacing an occurrence of a symbol $B \in N$ by a right-hand side of a production $p \in \bar{P}$. In the following we construct a $M$-controlled $EU$-accepting $t$-computation tree $T$ of $\underline{A}$. We have three cases to consider corresponding to (1)-(3) in Definition 4.2.

(i) $p = B \to \sigma(b)$ as in (1). Then $t = t'(B \leftarrow \sigma(b))$. [In $t'(B \leftarrow \sigma(b))$ $B$ denotes actually a fixed leaf that is labeled by $B$, this should not cause confusion.] Now in the computation of $T$, $\underline{A}$ reads first $b$ making the existential choice $\{\bar{B}\}$ by (4). This leads to the configuration $K_1 = t'(B \leftarrow \sigma(\bar{B}))$. Because $t' \in F_\Omega$, we have

$$yd(K_1) \in (V \cup N)^* \bar{B} (V \cup N)^* \subseteq M_1.$$

Hence $K_1 \in M$. In $K_1$ $\underline{A}$ reads the active subtree $\sigma(\bar{B})$ by (6) and the resulting configuration is $t'(B \leftarrow B) = t'$. By the induction assumption $t' \in H_{EU}(\underline{A})[M]$ and hence the computation tree $T$ can be completed from $t'$ to a $M$-controlled $EU$-accepting computation tree.

(ii) $p = B \to \sigma(\tau(C, D))$ as in (2). Now $t = t'(B \leftarrow \sigma(\tau(C, D)))$ and at the root of $T$ $\underline{A}$ reads the active subtree $\tau(C, D)$ of $t$ by rule (8) making the existential choice $\{\bar{B}\}$. This leads to an identical configuration $K_1$ as in case (i).

(iii) $p = B \to^D \sigma(\sigma(C))$ as in (3). Since $p$ can be applied to the leaf $B$ of $t'$, it follows that the leaf immediately to the left of $B$ is labeled by $D$. Now $t = t'(B \leftarrow \sigma(\sigma(C)))$ and in $T$ $\underline{A}$ reads the active subtree $\sigma(C)$ by (5) making the existential choice $\{(B, C)\}$. [$\{(B, C)\} \in \sigma_g(C)$ because $DB \to DC \in P$.] This leads to a configuration $K_2 = t'(B \leftarrow \sigma((B, C)))$, where

$$yd(K_2) \in (V \cup N)^* D (B, C) (V \cup N)^*.$$

Now $K_2 \in M$ since $DB \to DC \in P$. In $K_2$ $\underline{A}$ reads the active subtree $\sigma((B, C))$ by rule (7) and this leads to $t' \in H_{EU}(\underline{A})[M]$.

Thus we have shown that $D(G) \subseteq H_{EU}(\underline{A})[M]$. For the converse part of (11) suppose that $t \in H_{EU}(\underline{A})[M] \cap F_\Sigma(N)$. Suppose that

$$t\,(\vdash_{\underline{A}}^{EU})^n\, T$$

where $T \in \mathrm{COM}_{EU}(\underline{A}, t)[M]$ is an accepting computation tree. Since the recognizer $\underline{A}$ is "a nondeterministic automaton" (i.e., none of the rules causes universal branching in the computation tree), $T$ is in fact a computation path of length $n+1$. We show that $t \in D(G)$ using induction on $n$.

If $n = 0$, then $t \in A' \cap M = \{S\} \subseteq D(G)$. Then suppose that $n = k+1$ and that the claim holds for all values $n \leq k$. Let $K$ be the configuration that is obtained from $t$ in the computation tree $T$ by applying one of the rules (4), (5) or (8). [Rules (6) and (7) cannot be applied because $t \in F_\Sigma(N)$.]

As an example we consider the case where $K$ is obtained from $t$ using the rule (5); the two other cases are similar but simpler.

Let $f=\sigma(B)$ be the active subtree of $t$ such that $K=t(f \leftarrow (E, B))$, $E \in N$. Let $X \in V \cup N$ be the label of the leaf immediately to the left of $f$. (Since $K \in M$, there exists a leaf to the left of $f$.) Now

$$yd(K)=w_1 X(E, B) w_2,$$

where $w_1, w_2 \in (V \cup N)^*$. Since $T$ is $M$-controlled, it follows that $yd(K) \in M_1$ and hence $X$ is an element of $N$ such that $XE \rightarrow XB \in P$. Since the configurations of $M$ can contain at most one element of $\bar{N} \cup (N \times N)$, in the computation path $T$ the recognizer $\underline{A}$ must next "destroy" the state $(E, B)$ by rule (7). This means that in $t$ the father of $f$ must be labeled by $\sigma$ and in the computation path $T$ the successor of $K$ is

$$K'=t(\sigma(f) \leftarrow E) \in F_\Sigma(N).$$

Since $XE \rightarrow XB \in P$, it follows by (3) that $E \rightarrow^X \sigma(\sigma(B)) \in \bar{P}$ and hence

$$K' \vdash_G t.$$

Now there exists a $M$-controlled $EU$-accepting $K'$-computation path of length $k-1$ and hence by the induction assumption $K' \in D(G)$. Thus we have completed the proof of (11).

Now

$$L_{EU}(\underline{A})[M]=H_{EU}(\underline{A})[M] \cap F_\Sigma=D(G) \cap F_\Sigma$$

and hence by Lemma 4.3 for every $w \in V^+$:

$S \Rightarrow_G^* w$ iff there exists a $t \in L_{EU}(\underline{A})[M]$ such that $yd(t)=w$.

Thus

$$yd(L_{EU}(\underline{A})[M])=L(G)$$

and $L_{EU}(\underline{A})$ $[M] \in \mathscr{L}_{EU}(AR)$ by Theorem 3.3. □

*Example* 4.5: Consider the PNF grammar $G=(V, N, S, P)$, where

$$V=\{b, c, d\},$$

$$N=\{S, B, B_1, B_2, C_1, C_2, D_1, D_2, E, F\}$$

and the set $P$ consists of the following productions:

$$S \to ED_1,$$
$$E \to BF, \qquad E \to c,$$
$$F \to EB_1, \qquad F \to EB_2,$$
$$B_i B_j \to B_i B_i \quad \text{if} \quad \{i, j\} = \{1, 2\},$$
$$B_i D_j \to B_i C_i \quad \text{if} \quad \{i, j\} = \{1, 2\},$$
$$C_i \to D_i D_i, \qquad 1 \leqq i \leqq 2,$$
$$B \to b,$$
$$B_i \to b, \qquad 1 \leqq i \leqq 2,$$
$$D_i \to d, \qquad 1 \leqq i \leqq 2.$$

Now it can easily be verified that

$$L(G) = \{ b^n cb^n d^m \mid n \geqq 0, 1 \leqq m \leqq n+1 \}.$$

Note that $L(G)$ is not context-free. The word $bcbd^2$ is derived for instance as follows:

$$S \Rightarrow_G ED_1 \Rightarrow_G BFD_1 \Rightarrow_G BEB_2D_1$$
$$\Rightarrow_G BEB_2C_2 \Rightarrow_G BEB_2D_2D_2 \Rightarrow_G^* bcbd^2.$$

The derivation tree $t \in D(G)$ corresponding to this derivation is

$$t = \sigma(\tau(\sigma(\tau(\sigma(b), \sigma(\tau(\sigma(c), \sigma(b))))), \sigma^3(\tau(\sigma(d), \sigma(d))))).$$

Let the recognizer $\underline{A}$ and the control-forest $M$ corresponding to the grammar $G$ be as in the proof of Lemma 4.4. We denote $\vdash_{\underline{A}}^{EU}$ simply by $\vdash$. In the following we compute an $EU$-accepting $M$-controlled $t$-computation tree of $\underline{A}$. The computation trees of $\underline{A}$ have only one branch and therefore we denote a computation tree simply by the configuration that labels its single leaf:

$$t(\vdash)^{10} \sigma(\tau(\sigma(\tau(B, \sigma(\tau(E, B_2)))), \sigma^3(\tau(D_2, D_2))))$$
$$\vdash \sigma(\tau(\sigma(\tau(B, \sigma(\tau(E, B_2)))), \sigma^3(\bar{C}_2)))$$
$$\vdash \sigma(\tau(\sigma(\tau(B, \sigma(\tau(E, B_2)))), \sigma^2(C_2)))$$
$$\vdash \sigma(\tau(\sigma(\tau(B, \sigma(\tau(E, B_2)))), \sigma((D_1, C_2))))$$
$$\vdash \sigma(\tau(\sigma(\tau(B, \sigma(\tau(E, B_2)))), D_1))$$
$$\vdash \sigma(\tau(\sigma(\tau(B, \sigma(\bar{F}))), D_1))$$

$$\vdash \sigma(\tau(\sigma(\tau(\mathrm{B}, \mathrm{F})), \mathrm{D}_1))$$
$$\vdash \sigma(\tau(\sigma(\bar{E}), D_1))$$
$$\vdash \sigma(\tau(E, D_1)) \vdash \sigma(\bar{S}) \vdash S.$$

Now we get the main result as a direct consequence of Lemma 4.4.

THEOREM 4.6: *Every context-sensitive language can be effectively represented as an alternating yield-language.*

Since the proof of Theorem 2.6 is effective and the emptiness question for context-sensitive languages is undecidable (*cf.* [9]), it follows that:

COROLLARY 4.7: *Let* $Z \in \{EU, UE\}$. *The question whether for a given recognizer* $\underline{A} \in AR$, $L_Z(\underline{A}) = \varnothing$, *is undecidable.*

The above result was proved differently in [10]. Also, since the indexed languages form a proper subfamily of the context-sensitive languages (*cf.* [1]), one has

COROLLARY 4.8: *The family of yield-languages of algebraic forests is properly contained in the family of alternating yield-languages.*

The question whether all alternating yield-languages are context-sensitive remains open. However, here we see that at least one cannot effectively construct a context-sensitive grammar for a given alternating yield-language.

THEOREM 4.9: *The membership problem for alternating yield-languages is undecidable.*

*Proof.* (Outline): We reduce the Post Correspondence Problem, PCP, (*cf.* [4, 9]) to the above question. Let $V$ be an alphabet and let $\alpha = (\alpha_1, \ldots, \alpha_n)$, $\beta = (\beta_1, \ldots, \beta_n)$, $n \geqq 1$, $\alpha_i$, $\beta_i \in V^+$, $i = 1, \ldots, n$, be an arbitrary instance of PCP. Choose $\Sigma = \Sigma_2 \cup \Sigma_1 \cup \Sigma_0$, where $\Sigma_2 = \{\tau\}$, $\Sigma_0 = \{\omega_1, \omega_2\}$ and

$$\Sigma_1 = V \cup \{x(i) \mid x \in V, 1 \leqq i \leqq n\}.$$

Define the homomorphisms $h_1 : \Sigma_1^* \rightarrow V^*$ and $h_2 : \Sigma_1^* \rightarrow \{1, \ldots, n\}^*$ by the following:

$$h_1(x) = h_1(x(i)) = x,$$
$$h_2(x) = \lambda, \qquad h_2(x(i)) = i,$$

when $x \in V$, $i \in \{1, \ldots, n\}$.

We say that a $\Sigma$-tree $t$ is *well formed* if

(12) $t = \tau(t_1, t_2)$, where $t_1 = y_1 \ldots y_r(\omega_1)$ and $t_2 = z_1 \ldots z_s(\omega_2)$,

$r, s \geqq 1$, $y_i$, $z_j \in \Sigma_1$, and $y_1 \ldots y_r$ (resp. $z_1 \ldots z_s$) is obtained by catenating strings of the form $x_1(i) x_2 \ldots x_k$, $k \geqq 1$, $x_j \in V$, $j=1, \ldots, k$, $i \in \{1, \ldots, n\}$, and $\alpha_i = x_1 \ldots x_k$ (resp. $\beta_i = x_1 \ldots x_k$). (Intuitively this means that $t_1$ codes a string of words of $\alpha$ and $t_2$ a string of words of $\beta$.)

We define the forest $L$ to consist of all well formed trees $\tau(t_1, t_2)$ as in (12) such that

(13) $h_1(y_1 \ldots y_r) = h_1(z_1 \ldots z_s)$ (and hence also $r=s$), and

(14) $h_2(y_1 \ldots y_r) = h_2(z_1 \ldots z_s)$.

We construct an $AR$-recognizer $\underline{A}$ such that $L_{EU}(\underline{A}) = L$. When $\underline{A}$ reads a leaf $\omega_1$ of an input tree $t$, the computation branches universally into three computations $C_1$, $C_2$ and $C_3$. The computation $C_1$ just checks that the input tree $t$ is well formed, i.e., of the form $\tau(t_1, t_2)$ as in (12). This is easy since the set of well formed trees is regular.

Using a construction essentially similar to that of Appendix 2 of [10], in one branch of the computation $C_2$, $\underline{A}$ can be forced to read alternately unary symbols $y_i$ and $z_i$ from $t_1$ and $t_2$, and send along another branch of the computation tree "signals corresponding to $y_i$ and $z_i$" to the root $\tau$. Thus $\underline{A}$ is able to check that $r=s$ and $h_1(y_i) = h_1(z_i)$, $i=1, \ldots, r$. Similarly in $C_3$, $\underline{A}$ can check that $h_2(y_1 \ldots y_r) = h_2(z_1 \ldots z_s)$. [In $C_3$, $\underline{A}$ passes by deterministically symbols labeled by elements of $V$ and reads alternately from $t_1$ and $t_2$ symbols of the form $x(i)$, $x \in V$.] We leave the details of the construction to the reader. Instead of the above outlined construction from [10] one could also use a suitable regular control-forest to guarantee that $C_2$ and $C_3$ check conditions (13) and (14).

Now there exists a well formed $\Sigma$-tree that satisfies (13) and (14) iff the PCP $(\alpha, \beta)$ has a solution. An algorithm $M$ for the membership problem could decide whether $\omega_1 \omega_2$ belongs to $yd(L_{EU}(\underline{A}))$ and hence also whether $(\alpha, \beta)$ has a solution. Since PCP is undecidable, it follows that $M$ does not exist. □

From the proof of Theorem 4.9 it follows in fact that also the question whether a word of length two belongs to an alternating yield-language is undecidable.

THEOREM 4.10: *There does not exist an algorithm $M$ that for a given alternating yield-language $L$ would produce a context-sensitive grammar generating $L$.*

*Proof*: The membership problem for context-sensitive languages is decidable, *cf.* [9]. Hence if $M$ would exist, we could decide the membership problem for alternating yield-languages, contradicting Theorem 4.9. □

## 5. ALTERNATING EXTENDED YIELD-LANGUAGES

In the previous section we saw that using an alternating bottom-up computation we can recognize the derivation trees of a context-sensitive grammar that has no length decreasing productions. Here we consider extended yield-languages where some nullary symbol $c$ corresponds to the empty word in the yield. By labeling some leaves of the input tree with $c$ the recognizer can "extend its workspace" and thus it can recognize the derivation trees of arbitrary context-sensitive grammars.

DEFINITION 5.1: A four-tuple $G=(V, N, S, P)$ is called an *extended PNF grammar* if $G$ is otherwise exactly as in Definition 4.1 except that $P$ may also contain productions of the form

$$B \rightarrow \lambda,$$

where $B \in N$.

The following result is proved in [7].

THEOREM 5.2: *Every recursively enumerable language can be generated by an extended PNF grammar.*

Next we define the derivation trees of an extended PNF grammar analogously with Definition 4.2.

DEFINITION 5.3: Let $G=(V, N, S, P)$ be an extended PNF grammar. Corresponding to $G$ we define the ranked alphabet $\Omega=\Omega_2 \cup \Omega_1 \cup \Omega_0$, where $\Omega_0=V \cup N \cup \{c\}$, $\Omega_1=\{\sigma\}$ and $\Omega_2=\{\tau\}$. Here $\sigma$, $\tau$ and $c$ are new symbols.

The production set $\bar{P}$ is defined by rules (1)-(4), where (1), (2) and (3) are exactly as in Definition 4.2 and (4) corresponds to the additional $\lambda$-productions:

(4) If $B \rightarrow \lambda \in P$, then $B \rightarrow \sigma(c) \in \bar{P}$.

The set $\bar{P}$ defines the relation $\vdash_G$ and the set of derivation trees $D(G)$ as in Definition 4.2.

Analogously with Lemma 4.3 we have

LEMMA 5.4: *Let* $G=(V, N, S, P)$ *be an extended* PNF *grammar and* $w \in (V \cup N)^*$. *Then*

$S \Rightarrow_G^* w$ *iff there exists a* $t \in D(G)$ *such that* $yd_c(t)=w$.

LEMMA 5.5: *Let* $G=(V, N, S, P)$ *be an extended* PNF *grammar. Then there exists a ranked alphabet* $\Sigma$, $c \in \Sigma_0$ *and a* $\Sigma$*-forest* $M \in \mathscr{L}_{EU}(AR)$ *such that*

$$yd_c(M)=L(G).$$

*Proof.* — The proof is quite similar to the proof of Lemma 4.4. The $\lambda$-productions are treated in exactly the same way as the productions of the form $B \rightarrow b$. □

Since of course every alternating extended yield-language is recursively enumerable, we have as a consequence of Lemma 5.5:

THEOREM 5.6: *The family of alternating extended yield-languages equals to the family of recursively enumerable languages.*

ACKNOWLEDGMENTS

I would like to thank Magnus Steinby for many useful suggestions to improve the results and presentation of this paper. The grammar of Example 4.5 is due to Martti Penttonen.

## REFERENCES

1. A. AHO, *Indexed Grammars. An Extension of Context-free Grammars*, J. Assoc. Comput. Mach., Vol. 15, 1968, pp. 647-671.
2. A. K. CHANDRA, D. C. KOZEN and L. J. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., Vol. 28, 1981, pp. 114-133.
3. F. GÉCSEG and M. STEINBY, *Tree Automata*, Akadémiai Kiadó, Budapest, 1984.
4. J. E. HOPCROFT and J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
5. R. E. LADNER, R. J. LIPTON and L. J. STOCKMEYER, *Alternating Pushdown and Stack Automata*, SIAM J. Comput., Vol. 13, 1984, pp. 135-155.
6. D. E. MULLER, A. SAOUDI and P. E. SCHUPP, *Alternating Automata, the Weak Monadic Theory of the Tree, and its Complexity*, Proc. of 13th ICALP, Lect. Notes Comput. Sci., Vol. 226, 1986, pp. 275-283.
7. M. PENTTONEN, *One-sided and Two-sided Context in Formal Grammars*, Inf. Control, Vol. 25, 1974, pp. 371-392.
8. W. C. ROUNDS, *Mappings and Grammars on Trees*, Math. Systems Theory, Vol. 4, 1970, pp. 257-287.
9. A. SALOMAA, *Formal Languages*, Academic Press, New York, 1973.
10. K. SALOMAA, *Alternating Bottom-up Tree Recognizers*, Proc. of 11th CAAP, Lect. Notes Comput. Sci., Vol. 214, 1986, pp. 158-171.
11. G. SLUTZKI, *Alternating Tree Automata*, Proc. of 8th CAAP, Lect. Notes Comput. Sci., Vol. 159, 1983, pp. 392-404.