Volume 11 Number 4

ACTA CYBERNETICA

Szeged, 1994

Information for authors: Acta Cybernetica publishes only original papers in English in the field of computer science. Review papers are accepted only exceptionally. Manuscripts should be sent in triplicate to one of the Editors. The manuscripts must be typed douple-spaced on one side of the paper only. For the form of references, see one of the articles previously published in the journal.

Acta Cybernetica, Vol. 11, No. 4, Szeged, 1994

Codes and infinite words*

J. Devolder † M. Latteux† I. Litovsky‡ L. Staiger§

Abstract

Codes can be characterized by their way of acting on infinite words. Three kinds of characterizations are obtained. The first characterization is related to the uniqueness of the factorization of particular periodic words. The second characterization concerns the rational form of the factorizations of rational words. The third characteristic fact is the finiteness of the number of factorizations of the rational infinite words. A classification of codes based on the number of factorizations for different kinds of infinite words is set up. The obtained classes are compared with the class of ω-codes, the class of weakly prefix codes and the class of codes with finite deciphering delay. Complementary results are obtained in the rational case, for example a necessary and sufficient condition for a rational ω-code to have a bounded deciphering delay is given.

Résumé: La factorisation des mots infinis permet de caractériser les codes parmi les langages de mots finis. Les critères obtenus sont de trois types. Le premier critère est relatif à l'unicité de la factorisation de certains mots périodiques. Le second concerne la forme des factorisations des mots rationnels. Finalement, seuls les codes nous assurent de la finitude du nombre de factorisations des mots rationnels. Les codes sont classifiés selon le nombre de factorisations de certains types de mots infinis. Les classes obtenues sont étudiées et comparées avec les classes déjà définies de ω-codes, de codes faiblement préfixes et de codes à délai borné. Des résultats complémentaires sont obtenus dans le cas rationnel, en particulier il est donné une condition nécessaire et suffisante pour qu'un ω-code rationnel soit à délai borné.

Introduction

Codes, which are defined as the bases of free submonoids of monoids of (finite) words [1] were initially introduced by Schützenberger [19] in 1955. Since then, the

*This work has been partially supported by the PRC "Mathématiques et Informatique" and by the EBRA working group n° 3166 ASMICS.

†CNRS URA 369, LIFL, Université de Lille I, 59655 Villeneuve d'Ascq cedex, France.

‡LABRI, Université de Bordeaux I, ENSERB 33405 Talence cedex, France.

§Lehrstuhlinformatik II, Universität Dortmund, Postfach 500500, 4600 Dortmund 50, Deutschland.

study of some classes of codes, specially from the point of view of an easy decoding, has been very active. Here we study codes, and classes of codes from the particular point of wiew of decoding infinite words. In this respect, the interesting codes are those for which every infinite word has at most one factorisation. We shall refer to these codes as ω-codes. It was shown by Levenshtejn [12] that, for a finite code, any infinite word has at most one factorization iff this code has a bounded deciphering delay. For infinite codes the situation is more complicated. It turns out that the class of ω-codes (initially called ifl-codes by Staiger) properly contains the class of codes having a finite deciphering delay, which in one's turn properly contains the class of codes having a bounded deciphering delay [20]. The most interesting codes are codes with bounded deciphering delay, because they allow an easy decoding of finite and infinite words. We give at the end of this paper an interesting necessary and sufficient condition for a rational ω-code to have a bounded deciphering delay.

Although arbitrary codes may give several factorizations of infinite words, codes can be characterized by their way of acting on infinite words. This is the purpose of the first section. Indeed, a language C is a code if and only if, for every word v of C^+, the periodic infinite word v^ω has a single factorization over C. Codes are also characterized by the form of the factorizations of ultimately periodic words, and also by the fact that the number of factorizations of an arbitrary ultimately periodic word is finite. As an application, it is shown that the usual notion of code with bounded deciphering delay coincide with the notion defined in [20].

So, codes and ω-codes are characterized in terms of infinite words. It is obvious that a language C is a code if no infinite word has uncountably many factorizations over C. Having this fact in mind, we set up a classification of codes based on the number of factorisations for different kinds of infinite words. If C denotes a code, the kinds of infinite words that we consider are the following ones: periodic words of the form u^ω with $u \in C^+$, periodic words, ultimately periodic words and any infinite words. This leads to consider the class $\mathbf{C}$ of codes, the class Π of π-codes, the class $\mathbb{W}$ of weakly prefix codes, the class $\mathbf{I}$ of ω-codes (I as "ifl code").

These classes are compared with each other, and also compared with the class $\mathbb{B}$ of codes having a bounded deciphering delay, the class $\mathbb{D}$ of codes having a finite deciphering delay, the class $\mathbb{V}$ of circular codes ($\mathbb{V}$ as "very pure"), the class $\mathbf{S}$ of suffix codes. The results can be summarized by the following strict inclusions $\mathbb{B} \subset \mathbb{D} \subset \mathbf{I} \subset \mathbb{W} \subset \Pi \subset \mathbf{C}$, $\mathbb{V} \subset \mathbb{W}, \mathbf{S} \subset \Pi$, and by the next array which indicates the maximal number of factorizations according to the type of infinite words and the class of codes, when the alphabet is countable and has at least two elements. In this array, the stars $\star$ point out the characteristic properties, and ∞ denotes Card($I\!R$) : a noncountable infinity of factorizations is possible.

words	u^ω ($u \in C^+$)	u^ω	uv^ω	any
languages				
ω-codes	1	1	1	$1\star$
weakly prefix	1	1	$1\star$	∞
π-codes	1	$1\star$	finite	∞
codes	$1\star$	finite $\star$	finite $\star$	∞

In the second section, we give characterizations for the classes $\mathbb{W}$,Π and $\mathbf{S}$ and we prove the announced inclusions. Using the inclusions $\mathbb{V} \subset \mathbb{W}$,$\mathbf{S} \subset \Pi$ and the composition of codes, one can easily construct ω-codes, weakly prefix codes and π-codes. The second section terminates by some examples which enable us to fulfill the array.

In the last section, we examine the modifications holding when C is a rational language. Every infinite word has then a finite bounded number of C-factorizations whenever C is a code. The notion of ω-code coincides with the notion of weakly prefix code in the rational case. We give also a new interesting necessary and sufficient condition for a rational ω-code to have a finite deciphering delay. This condition $C^\omega \cap C^* \text{ Adh}(C) = \emptyset$ can be easily checked. As expected, it is decidable whether a rational language belongs to any class $\mathbb{B}, \mathbb{I}$ (i.e. $\mathbb{W}$) or Π.

Notations and basic definitions:

In the following, we consider an alphabet (finite or not) A, the set A^* (resp. A^ω) of all finite (resp. infinite) words over A, the set A^+ which denotes the language $A^* - \varepsilon$, where ε is the empty word. The length of a word u is denoted by $|u|$. The symbol $\leq$ (resp $<$) denotes the relation between words "is a (resp. strict) prefix of". The left quotient of a word u by a word v is denoted by $v^{-1}u$.

Two words x and x' are said to be *conjugate* if there exist u and v such that $x = uv$ and $x' = vu$. A word $z \in A^+$ is *primitive* if $z = u^n$ implies $n = 1$.

Given a language $C \subset A^+$, the submonoid generated by C is the language $C^* = \{v_1 \dots v_n | n \geq 0, v_i \in C, 1 \leq i \leq n\}$ and C^ω stands for the set of infinite words obtained by concatenation of an infinite sequence of words of C : $C^\omega = \{v_0 v_1 v_2 \dots | v_i \in C, i \geq 0\}$. A *$C$-factorization* of a word $v \in C^*$ is a sequence of words of C : $(v_1, \dots, v_n)$ such that $v = v_1 \dots v_n$. A *C-factorization* of a word $v \in C^\omega$ is a sequence of words of C : $(v_0, v_1, v_2, \dots)$ such that $v = v_0 v_1 v_2 \dots$.

An infinite word w is said to be *ultimately periodic* if there finite words u and v such that $w = uv^\omega$. It is said to be *periodic* if u can be chosen equal to ε.

Given a language $C \subset A^+$ we shall often consider a bijection φ between an alphabet X and the language C. This mapping can be extended to X^* as a morphism $\varphi : X^* \to C^*$. This morphism is said to be a *coding morphism for C* (even if it is not injective). The mapping φ can also be extended to X^ω ($\varphi(z_0 z_1 \dots)$ is the word $\varphi(z_0)\varphi(z_1)\dots$). These extension agree with the composition of functions of words of X^* (resp. X^ω) and the set of C-factorizations of words of C^* (resp C^ω). Thus a C-factorization of $u \in C^*$ (resp : $u \in C^\omega$) will be represented by an element of X^* (resp: X^ω).

Definitions: Let C be a language $\subset A^+$.

- C is a code if and only if $\forall u, v \in C \quad uC^* \cap vC^* \neq \emptyset \Rightarrow u = v$
- C is a prefix code if and only if $\forall u, v \in C \quad u \leq v \Rightarrow u = v$
- C is an ω-code if and only if $\forall u, v \in C \quad uC^\omega \cap vC^\omega \neq \emptyset \Rightarrow u = v$ [20].

These definitions can be expressed in terms of morphisms. Let φ be any coding morphism for C.

- C is a code if and only if $\varphi : X^* \to C^*$ is injective.
- C is an ω-code if and only if $\varphi : X^\omega \to C^\omega$ is injective.

Recall that ω-codes are codes and that prefix codes are ω-codes. Using coding morphisms, it is easily seen that a composition theorem holds for codes [1] and ω-codes. Namely, let C be a language $\subset X^+$ and $\varphi : X^* \to A^*$ be a coding morphism for a language $D = \varphi(X) \subset A^+$, if C and D are codes (resp: ω-codes), $\varphi(C)$ is a code (resp: ω-code).

1 Characterizations of codes

In this section, three kinds of characterisations for codes are obtained: the first kind concerns the words which have only one C-factorisation, the second is related to the form of the C-factorisations of ultimately periodic words, the last give a bound for the number of C-factorisations of a given ultimately periodic word.

We define now some notations and give some lemmata used in the proof of the main theorem. Let $\varphi : X^* \to C^*$ be a coding morphism for C.

Lemma 1.1 *If $C \subset A^+$ is a code, for every word $v \in A^+$, there exists at most one primitive word $z \in X^+$ such that $\varphi(z) \in v^+$.*

Proof. If $\varphi(z) = v^n$ and $\varphi(z') = v^m, \varphi(z^m) = \varphi(z'^n)$. Thus $z^m = z'^n$ (φ injective) and then $m = n$ and $z = z'$ if z and z' are primitive words. □

Lemma 1.2 *If $yz^\omega \in X^\omega$ is a C-factorization of uv^ω (where v is assumed to be a primitive word), $\varphi(z)$ is a power of a conjugate of v.*

Lemma 1.3 *Let us consider $x \in X^\omega$ such that $\varphi(x)$ is ultimately periodic. There exist $y, z \in X^*, t \in X^\omega$ such that $x = yzt$, and $\varphi(x) = \varphi(y)\varphi(z)^\omega$.*

Proof. Let x be the C-factorization: $u_1, u_2, \ldots, u_p, \ldots$ of an ultimately periodic word uv^ω. Since v is of finite length there exist i, j, k, m such that $k < m$, $u_1 \ldots u_k = uv^i w$ and $u_1 \ldots u_m = uv^{i+j} w$ where w is a prefix of v. The word $v' = w^{-1}v^j w$ belongs to C^+ and $uv^\omega = u_1 \ldots u_k v'^\omega$. □

Lemma 1.4 *If $C \subset A^+$ is a code, for every word $v \in C^+$, the word v^ω has only one C-factorization.*

Proof. Let us consider $v \in C^+ : v = v_1 v_2 \ldots v_n$ with $v_i \in C$ such that v^ω has two distinct C-factorisations: $v^\omega = (v_1 v_2 \ldots v_n)^\omega = u_1 u_2 \ldots u_p \ldots$ (where $\forall i\ u_i \in C$). Without loss of generality we may assume that $v_1 \neq u_1$. As in the proof of lemma 1.3, there exist i, j, k, m such that $k < m$, $u_1 \ldots u_k = v^i w$ and $u_1 \ldots u_m = v^{i+j} w$ where w is a prefix of v. Then the word $v^{i+j} w = u_1 \ldots u_m = (v_1 \ldots v_n)^j u_1 \ldots u_k$ has two distinct C-factorizations. C is not a code. □

Lemma 1.5 *Consider $C \subset A^+$ such that every word of the form w^ω with $w \in C^+$ has exactly one C-factorization. For all words $u, v \in A^+$, every C-factorization of the word uv^ω is ultimately periodic.*

Proof. Let us consider a C-factorization x of the word $uv^\omega \in C^\omega$. From lemma 1.3, there exist $y, z \in X^*, t \in X^\omega, v' \in C^+$ such that $x = yzt, \varphi(z) = v', \varphi(t) = v'^\omega$. By hypothesis, the word v'^ω has a single C-factorization. Since $\varphi(z^\omega) = v'^\omega = \varphi(t)$, we have $t = z^\omega$ and then x is ultimately periodic. □

Lemma 1.6 *Let C be a code $\subset A^+$. Consider words u and v of A^+. The set of C-factorizations of uv^ω is finite.*

Proof. Let us consider $uv^\omega \in C^\omega$. Assume that v is primitive. Denote by $V = \{v_i | i \in I\}$ the set of conjugates v_i of v such that $v_i^+ \cap C^+ \neq \emptyset$. Since C is a code, we can denote by z_i the primitive word such that $\varphi(z_i) \in v_i^+$ and n_i the corresponding power of v_i : $\varphi(z_i) = v_i^{n_i}$. We consider the equivalence relation on V:

$$v_i \simeq v_j \Leftrightarrow z_i \text{ and } z_j \text{ are conjugate.}$$

Since $\varphi(z_i)$ and $\varphi(z_j)$ are conjugate, it is clear that $n_i = n_j$ whenever $v_i \simeq v_j$.

Let F be the set of C-factorizations of uv^ω. We shall prove that $\mathrm{Card}(F) \leq \Sigma n_i$, where only one n_i by $\simeq$ class is taken.

Since C is a code, from lemma 1.5, every C-factorization of uv^ω is ultimately periodic, hence of the form yz^ω with z primitive; from lemma 1.2, there exists a conjugate v_i of v such that $\varphi(z) \in v_i^+$. Then the set F of C-factorizations of uv^ω satisfies $F = \cup(F \cap X^* z_i^\omega)$. Since $X^* z_i^\omega = X^* z_j^\omega$ when z_i and z_j are conjugate, the previous union has only N terms, where N denotes the number of classes of $\simeq$.

It remains to prove that $\mathrm{Card}(F \cap X^* z_i^\omega) \leq n_i$. Consider $y' z_i^\omega, y'' z_i^\omega$ and $y z_i^\omega \in F$, such that $|\varphi(y)| = \inf\{|\varphi(u)| | u z_i^\omega \in F\}$. Since $\varphi(y)\varphi(z_i)^\omega = \varphi(y')\varphi(z_i)^\omega, \varphi(z_i) \in v_i^+$ and v_i is primitive, one has $\varphi(y') = \varphi(y)v_i^{h'}$ for some h'. One has also $\varphi(y'') = \varphi(y)v_i^{h''}$ for some h''. If $h' = kn_i + h''$, $\varphi(y'' z_i^k) = \varphi(y)v_i^{h''+kn_i} = \varphi(y)v_i^{h'} = \varphi(y')$. Since C is a code $y'' z_i^k = y'$ and then $y'' z_i^\omega = y' z_i^\omega$. The number of elements of $F \cap X^* z_i^\omega$ is then at most the number n_i of integers modulo n_i. □

The following theorems give the characterization of codes. For convenience, theorem 1.7 gives the characterizations related to periodic words ant theorem 1.8 gives those related to ultimately periodic words.

Theorem 1.7 *Let C be a language $\subset A^+$. The following assertions are equivalent:*

1. *C is a code,*
2. *for every $u \in C^+, u^\omega$ has a single C-factorization,*
3. *every C-factorization of each periodic infinite word is ultimately periodic,*
4. *each periodic infinite word has a finite number of C-factorizations.*

Theorem 1.8 *Let C be a language $\subset A^+$. The following assertions are equivalent:*

1 *C is a code,*

3' *every C-factorization of each ultimately periodic infinite word is ultimately periodic,*

4' *each ultimately periodic infinite word has a finite number of C-factorizations.*

Proof. $1 \Rightarrow 2$: lemma 1.4; $2 \Rightarrow 3'$: lemma 1.5; $1 \Rightarrow 4'$: lemma 1.6; $3' \Rightarrow 3, 4' \Rightarrow 4$: clear; $3 \Rightarrow 1$ and $4 \Rightarrow 1$: If C is not a code, there exists a word u which has two distinct C-factorizations. There exist y and $z \in X^+, y \neq z$, such that $\varphi(y) = \varphi(z) = u$. Without loss of generality, one can assume that the first letters of y and z are different, then a bijection Ψ between $\{0,1\}$ and $\{y,z\}$ gives a bijective morphism $\Psi : \{0,1\}^\omega \to \{y,z\}^\omega$ and the elements of $\{y,z\}^\omega$ are distinct C-factorizations of u^ω. The word u belongs to C^+ and u^ω has a non-countable set of C-factorizations; hence also a non-countable number of non-ultimately periodic C-factorizations. □

Remarks:

– From lemma 1.3, in the property 4', one can replace: "each ultimately periodic infinite word" by "each ultimately periodic infinite word of the form uv^ω with $u, v \in C^+$".

– A periodic infinite word can have a nonperiodic C-factorization even if C is a (prefix) code. For example: if $C = \{a, ba\}$, the C-factorization of $(ab)^\omega$ is not periodic.

Property 3' of codes has been used to give characteristic properties of precircular codes [7]. The characterizations 3 and 3' can be used to prove composition theorems for weakly prefix codes and for π-codes. As an application of property 2, it can be easily seen that a code C is always minimal in the family of ω-generators of C^ω (*i.e.* languages R such that $R^\omega = C^\omega$). We give here another application of property 2.

Application:

In [20] the following notion of delay of decipherability was introduced: a language $C \subset A^+$ is said to have a *finite delay of decipherability* if

$$\forall v \in C \; \exists m(v) \geq 0 \quad vC^{m(v)}A^\omega \cap C^\omega \subset vC^\omega.$$

Remark: A language with a finite delay of decipherability in this sense is not necessarily a code, as it can be seen for $C = \{a, a^2\}$. The language $C = \{a^2, a^3, b\}$ is another more complicated example (it is not a code but $m(b) = 0$ and $m(a^2) = m(a^3) = 1$).

Some authors use another notion of finite deciphering delay [1], [5], which is in fact a notion of bounded deciphering delay [10]. Here, we say that:
– a language $C \subset A^+$ is said to have a *finite deciphering delay* if

$$\forall v \in C \quad \exists m(v) \geq 0 \quad \forall v' \in C \quad (vC^{m(v)}A^\omega \cap v'C^\omega \neq \emptyset \Rightarrow v = v')$$

or equivalently if

$$\forall v \in C \quad \exists m(v) \geq 0 \quad \forall v' \in C \quad (vC^{m(v)}A^* \cap v'C^* \neq \emptyset \Rightarrow v = v')$$

A language which has a finite deciphering delay is a code [1] and clearly has a finite delay of decipherability in the sense of [20]. Thus the notion of finite deciphering delay is stronger than the notion defined by Staiger. We shall see that these notions coincide for codes.

Proposition 1.9 *Every code which has a finite delay of decipherability is an ω-code.*

Proof. Consider $v, v' \in C$ such that $vC^\omega \cap v'C^\omega \neq \emptyset$. For $n \geq \max(m(v), m(v'))$ and $w \in vC^\omega \cap v'C^\omega$, there exist $u, u' \in C^n$ such that vu and $v'u'$ are prefixes of w. If vu is a prefix of $v'u'$, $(v'u')^\omega \in vC^{m(v)}A^\omega \cap C^\omega$, thus $(v'u')^\omega \in vC^\omega$. Since $v'u' \in C^+$, from characterization 2, $v = v'$. Hence C is an ω-code. □

Proposition 1.10 *Every code which has a finite delay of decipherability has a finite deciphering delay.*

Proof. Let v and $v' \in C$ and assume that $vC^{m(v)}A^* \cap v'C^*$ is not empty. Consider $w \in vC^{m(v)}A^* \cap v'C^*$. The word wv^ω belongs to $vC^{m(v)}A^\omega \cap C^\omega$ and then belongs to vC^ω and to $v'C^\omega$, from the previous proposition we obtain that $v = v'$.

□

Remarks:

– In a same way, a code satisfying: $\exists m \geq 0 \quad \forall v \in C \quad vC^{m(v)}A^\omega \cap C^\omega \subseteq vC^\omega$ is a code with a bounded deciphering delay, that is to say:

$$\exists m \geq 0 \forall v \in C \forall v' \in C (vC^m A^* \cap v'C^* \neq \emptyset \Rightarrow v = v').$$

– The two notions of finite and bounded delay do not coincide in general, although they are equivalent in the regular case [20].

– The notions of ω-code and code with a (finite or bounded) deciphering delay coincide in the case of finite codes [12] [5]; these classes do not coincide when regular codes are considered [20] . We give in section 3 a necessary and sufficient condition for a rational ω-code to have a finite deciphering delay.

2 Study of some special codes - examples.

Weakly prefix codes were defined by Capocelli [5]:

Definition: A code $C \subset A^+$ is a weakly prefix code if and only if

$$\forall u, v, w \in A^* \quad (w, wu, uv, vu \in C^* \Rightarrow u \in C^*).$$

Notice that this definition is equivalent to the next:

A language $C \subseteq A^+$ is a *weakly prefix code* if and only if C is the base of a monoid M satisfying the condition:

$$\forall u, v, w \in A^* \quad (w, wu, uv, vu \in M \Rightarrow u \in M).$$

Proof. It is sufficient to prove that a monoid M which satisfies the required condition is stable [1]. If the words w, wu, uv', v' belong to M, the words $w, wu, uv'w, v'wu$ belong also to M. Let $v = v'w$. The words w, wu, uv, vu belong to M and then u belong to M, M is stable. □

Clearly, prefix codes are weakly prefix codes.

Let us recall some definitions. A language $C \subset A^+$ is a circular code [11] [1] if and only if

$$\forall n, p \geq 0 \forall u_0, \ldots, u_{n-1}, v_0, \ldots, v_{p-1} \in C \forall t \in A^* \forall s \in A^+ \text{ such that } v_0 = ts$$

$$(u_0 \ldots u_{n-1} = sv_1 \ldots v_{p-1}t \Rightarrow n = p \;\; t = \varepsilon \text{ and } \forall i \;\; u_i = v_i).$$

A monoid $M \subset A^*$ is a very pure monoid if and only if

$$\forall u, v \in A^* \quad (uv, vu \in M \Rightarrow u, v \in M).$$

It is known that a language C is a circular code if and only if C is the base of a very pure monoid [16].

Clearly, the class **V** of circular codes is an interesting subclass of the class **W** of weakly prefix codes. But the inclusion $\mathbf{V} \subset \mathbf{W}$ is strict: for example, $\{ab, ba\}$ is a (weakly) prefix code but is not a circular code.

The next proposition characterizes weakly prefix codes in terms of infinite words.

Theorem 2.1 *Let C be a language $\subset A^+$. The following assertions are equivalent:*

1. *C is a weakly prefix code*
2. *for every $u, v \in C^+$ uv^ω has a single C-factorization*
3. *each ultimately periodic infinite word has at most one C-factorization.*

Proof. $1 \Rightarrow 2$: Notice that, since C is a code, any C-factorisation of an ultimately periodic word is ultimately periodic. Assume now that $uv^\omega = u'v'^\omega$ and $u, u', v, v' \in C^+, |u| \leq |u'|$. If the two C-factorisations are distinct, we can assume that $u = u_1 \dots u_n$ and $u' = u'_1 \dots u'_p$ with $u_1 \neq u'_1$. If $u' = uw$ where $w \in C^*, C$ is not a code. Then suppose that $u' = uw$ where $w \notin C^*$. Taking appropriate powers of v and v', we can assume that $|v| = |v'| > |w|$. Then $v = ww'$ and $v' = w'w$ for some word w'. We have $u, uw, ww', w'w \in C^*$ but $w \notin C^*$, a contradiction with C weakly prfix.

$2 \Rightarrow 1$: If C is not weakly prefix, there exist u, v, w such that $u \notin C^*, w, wu, uv, vu \in C^*$. Hence $w(uv)^\omega$ has two distinct C-factorizations.

$3 \Rightarrow 2$: Clear. $2 \Rightarrow 3$: Clear from lemma 1.3. □

As a consequence we obtain:

Corollary 2.2 *ω-codes are weakly prefix codes.*

The converse is not true in general. Let $C = \{ab\} \cup \{ab^n ab^{n+1} | n \geq 1\}$. This example presents a weakly prefix (circular) code C which is not an ω-code, but such that every proper subset of C is an ω-code. This shows a difference between ω-codes and weakly prefix codes since a language C is clearly a weakly prefix code iff every finite subset of C is a weakly prefix code. This example shows also that V and W are not included in the class I of ω-codes; I is neither included in V (consider the prefix code $\{ab, ba\}$).

Now, we study a type of codes which take place between codes and weakly prefix codes. Indeed, such a type of codes exists. From theorem 1.7, if C is a code, for every $u \in C^+ u^\omega$ has a single C-factorization. But it is not possible to replace "$u \in C^+$" by "$u \in A^+$". This observation was already made by Karhumäki in connection with theorem 3.3 of [10], however the example given there, $\{ab, aba, baba\}$ is not a code. By contrast, the language $C = \{a, aaba, abaaba\}$ is a code and the word $(aab)^\omega$ has two C=factorisations.

In theorem 2.1, it is not possible to replace "ultimately periodic" by "periodic": a language C may no longer be a weakly prefix code even if every periodic infinite word has at most one C-factorisation. For example, let $C = \{ab, aba, ba^2\}$. The word $ab(aba)^\omega = aba(ba^2)^\omega$ is the only word which has at least two C-factorizations beginning by two different words. Thus every periodic word has at most one C-factorisation. Note that C is a suffix code.

Thus theorem 1.7 and 2.1 do not study uniqueness of the factorization of periodic words. Then it is natural to try to characterize codes which factorize infinite periodic words in a single manner. For sake of convenience these codes are called *π-codes* here. Note that the three-element codes which are not π-codes have been studied by Karhumäki and called periodic codes [10].

Definition: A language $C \subset A^+$ is said to be a π-code if each periodic infinite word has at most one C-factorisation.

Theorem 1.7 ensures that a π-code is a code. We have seen an example showing that the converse is false. As for weakly prefix codes, a technical characterization

of π-codes can be obtained. One can prove that a code $C \subset A^+$ is a π-code if and only if C satisfies the property:

(P) $\forall u, v, w, \beta \in A^*$ such that $wuvu < \beta^\omega$ and $|u| \geq |\beta|$, one has:

$$w, wu, uv, vu \in C^* \Rightarrow u \in C^*.$$

Proof. Let u, v, w, β such that $wuvu < \beta^*, |u| \geq |\beta|$ and $w, wu, uv, vu \in C^*$. We can assume β primitive, then u has a single interpretation over β : there exist a single $i \geq 0$, a single suffix of β : β', a single prefix of β : β'' such that $u = \beta'\beta^i\beta''$. Then $uv = \beta'\beta^j\beta\beta'^{-1}$ and $vu = \beta''^{-1}\beta\beta^j\beta''$ for some j. Hence $\beta^\omega = w(uv)^\omega = wu(vu)^\omega$. Since C is a π-code, the word β^ω h as at most one C-factorisation therefore $u \in C^*$.

Conversely, let β^ω be a periodic word having two distinct C-factorisations: $(w_1, w_2, \ldots)$ and $(w'_1, w'_2, \ldots)$. We can assume that $w_1 \neq w'_1$. Denote $\beta^\omega = u_1u_2\ldots$ where $u_i = \beta$ for each i.

We can consider (when exists) p_i such that $w_1 \ldots w_{p_i-1} < u_1 \ldots u_{i-1} \leq w_1 \ldots w_{p_i} \leq u_1 \ldots u_i$. There exist a word α and infinitely many i such that $w_1 \ldots w_{p_i} = u_1 \ldots u_{i-1}\alpha$. In the sequel, m and n denote such indices p_i. In a same way, there exist a word α' and infinitely many i such that there exists q_i satisfying $w_1 \ldots w_{q_i-1} < u_1 \ldots u_{i-1} \leq w_1 \ldots w_{q_i} \leq u_1 \ldots u_i$ and $w'_1 \ldots w'_{q_i} = u_1 \ldots u_{i-1}\alpha'$. In the sequel, m' and n' denote such indices q_i.

Let us choose m, m', n, n' such that $w_1 \ldots w_m < w'_1 \ldots w'_{m'} < w_1 \ldots w_n < w'_1 \ldots w'_{n'}$. Let $w = w_1 \ldots w_m, wu = w'_1 \ldots w'_{m'}$, $wuy = w_1 \ldots w_n, wuyz = w'_1 \ldots w'_{n'}$. The choice of m' can be done such that $|u| \geq |\beta|$. We have: $uy = \beta'^p \in C^+$ and $yz = \beta''^p \in C^+$, where β' and β'' are conjugate with β. Let $v = y(uy)^{q-1}; uv \in C^+$ and $vu = (yz)^p$, then $vu \in C^+$. The words w, wu, uv, vu belong to C^*, therefore u belongs to C^*, which gives a contradiction with "C is a code" since $w_1 \neq w'_1$. □

In this characterisation, the condition "C is a code" cannot be suppressed. For example, let $C = \{ba, b, abc, bc\}$. The monoid C^* is not free and the condition (P) is satisfied.

From theorem 2.1, it is clear that weakly prefix codes are π-codes. Surprisingly, the family of π-codes contains a well-known subfamily: the family S of suffix codes. This fact is obtained as a consequence of the next interesting characterization of suffix codes.

Proposition 2.3 *A language $C \subset A^+$ is a suffix code if and only if every C-factorization of a periodic infinite word is periodic.*

Proof. If C is not a suffix code, there exist $v' \in A^+, u, v \in C$ such that $v = v'u$. The word uv^ω is periodic and has a non periodic C-factorization.

Conversely, consider a suffix code $C, \varphi : X^* \to C^*$ a coding morphism for C and β a primitive word such that $\beta^\omega \in C^\omega$. Consider a C-factorization of β^ω. From lemma 1.2 and theorem 1.7, this factorization can be written yz^ω and there exists a conjugate of $\beta : \beta'$ such that $\varphi(z) = \beta'^n$ for some n. Since $\beta^\omega = \sigma\beta'^\omega, \varphi(y) = \sigma\beta'^k$ for some k. Then $\varphi(y)$ is a suffix of $(\beta'^n)^+$, and since C is a suffix code, y is a suffix of z^+. Hence the considered factorization is periodic. □

In these conditions, $\beta^\omega = v^\omega$ for some v in C^+ and from theorem 1.7, the C-factorisation of β^ω is unique. So we have:

Corollary 2.4 *Suffix codes are π-codes.*

Remarks:

– The inclusion $\mathbb{S} \subset \Pi$ is strict: the π-code $\{a, ba\}$ is a not a suffix code.

– A code with a finite left deciphering delay (even delay 1) is not always a π-code.

For example: the word $(abc)^\omega$ has two C-factorisations when $C = \{a, ab, cab, bca\}$.

– $\mathbb{S}$ is not included in $\mathbb{W}$: $\{c, ca, aba, ba^2\}$ is a suffix code which is not weakly prefix.

As an application of theorems 1.8 and 2.1, a composition property for weakly prefix codes and π-codes can be obtained:

Proposition 2.5 *Let C be a language $\subset X^+$ and $\varphi : X^* \to A^*$ be a coding morphism for a language $D = \varphi(X) \subset A^+$. If C and D are weakly prefix codes, $\varphi(C)$ is a weakly prefix code. If C is a weakly prefix code and D a π-code, $\varphi(C)$ is a π-code.*

Remark: In proposition 2.5, for $\varphi(C)$ to be a π-code, the request property "C weakly prefix code" cannot be replaced by the other one "C π-code". For example, $C = \{c, ca, aba, baa\}$ is a π-code but not a weakly prefix code (the word $c(aba)^\omega$ has two C-factorisations). Let $\varphi(a) = ac, \varphi(b) = b, \varphi(c) = c$. The code $D = \{ac, b, c\}$ is prefix but $\varphi(C) = \{c, cac, acbac, bacac\}$ is not a π-code since the word $(cacba)^\omega$ has two C-factorisations.

In the following, we give some examples of π-codes and weakly prefix codes for which there exists a word w_0 which has infinitely many factorisations. The set of factorisations of w_0 may be countable or not countable. The last example allows us to fulfill the array given in the introduction.

Example 2.1 *Let $C_1 = \{aba^2b^2a^3b^3 \dots a^nb^na^{n+1} | n \geq 1\}$, $C_2 = \{b^pa^qb^p | 0 < p < q\}$ and consider $C = C_1 \cup C_2$. The language C is a suffix code and thus a π-code, but C is not a weakly prefix code since for example, the word $aba^2b^2a^3b^3(a^4b^4)^\omega$ has two C-factorizations*

The word $w_0 = aba^2b^2 \dots a^nb^na^{n+1}b^{n+1} \dots$ has a countable infinity of C-factorizations and every word has a countable (finite or not) number of C-factorizations.

Example 2.2 *Let $A = \{a, b\}, C = \{uab^n | |u| = n, n \geq 0, |u|_a = 0 \text{ or } 1\}$ ($|u|_a$ denotes the number of occurrences of a in u). Clearly C is a suffix code thus a π-code. Since the word $w = bab.bab.(b^4ab^4.bab)^\omega = bab^2ab^4.(bab.b^3ab^4)^\omega$ has two C-factorizations, C is not a weakly prefix code. We shall see that there exists a word w_0 which has a noncountable infinity of C-factorizations.*

Let w_0 be the word: $ab^{i_0}ab^{i_1} \dots ab^{i_n} \dots$ where $i_0 = 0, i_1 = 1, i_{n+2} = i_{n+1} + i_n + 1$ for every $n \geq 0$. Let us prove that, for every factorisation of w_0 : $w_0 = uv$, the word v has at least two C-factorizations. In fact, $v \in x(v)C^\omega \cap y(v)C^\omega$ for two different words: $x(v)$ and $y(v)$ of C. Let $v = b^{j_0}ab^{i_n}ab^{i_{n+1}} \dots$ with $0 \leq j_0 \leq i_{n-1}$. Then $v = b^{j_0}ab^{j_0}.b^{j_1}ab^{j_1} \dots .b^{j_h}ab^{j_h} \dots$ where $j_{h+1} = i_{n+h} - j_h$ and j_h satisfies $0 \leq j_h \leq i_{n+h}$ for every $h \geq 0$; let us set $x(v) = b^{j_0}ab^{j_0}$. The word v has also the other C-factorisation: $v = b^{j_0}ab^{i_n}ab^{k_0}.b^{k_1}ab^{k_1} \dots .b^{k_h}ab^{k_h} \dots$ where $k_0 = j_0 + i_n + 1, k_{h+1} = i_{n+1+h}k_h$ and k_h satisfies $0 \leq k_h \leq i_{n+1+h}$ for every $h \geq 0$; let us set $y(v) = b^{j_0}ab^{i_n}ab^{k_0}$. We have: $v \in x(v)C^\omega \cap y(v)C^\omega$.

Then an injective mapping δ form $\{0,1\}^{\mathbb{N}}$ into the set of C-factorizations of w_0 can be defined next way: let $\beta = (\beta_n)_n \in \{0,1\}^{\mathbb{N}}$, $\delta(\beta) = (z_n)_n$ where $z_0 = x(w_0)$

if $\beta_0 = 0$ and $z_0 = y(w_0)$ if $\beta_0 = 1, z_n = x((z_0 z_1 \dots z_{n-1})^{-1} w_0)$ if $\beta_n = 0$ and $z_n = y((z_0 z_1 \dots z_{n-1})^{-1} w_0)$ if $\beta_n = 1$. So w_0 has a noncountable infinity of C-factorisations. □

Example 2.3 *Let $A = \{u_i | i \geq 0\}$ and $C = C_1 \cup C_2$ where $C_1 = \{u_0 u_1 u_2 \dots u_{2^i} | i \geq 0\}$ and $C_2 = \{u_{2^i 3^j + 1} \dots u_{2^i 3^{j+1}} | i, j > 0\}$. Since the mapping: $(i, j) \mapsto 2^i 3^j$ is injective, it can be shown that C is a weakly prefix code. Every word has a countable (finite or not) number of C-factorizations and there exists a word which has a countable infinity of C-factorizations. Indeed the word $w_0 = u_0 u_1 u_2 \dots u_n \dots$ has a countable infinity of C-factorizations since the C-factorizations of w_0 are of the form: $(u_0 \dots u_{2^i})(u_{2^i+1} \dots u_{2^i 3}) (u_{2^i 3 + 1} \dots u_{2^i 3^2}) \dots (u_{2^i 3^j + 1} \dots u_{2^i 3^{j+1}}) \dots$ for some $i \geq 0$.* □

Example 2.4 *Let $A = \{u_i | \geq 1\}, C = \{u_n \dots u_{2n-1} | n \geq 1\} \cup \{u_n \dots u_{2n} | n \geq 1\}$. We show that C is a weakly prefix code such that there exists a word which has a noncountable infinity of C-factorizations.*

Let $w_0 = u_1 u_2 \dots u_n \dots$. As in example 2.2, it can be easily verified that w_0 has a noncountable infinity of C-factorizations. Let w be a word which has two C-factorizations δ and δ' beginning by two different words. Then δ and δ' begin by $u_n \dots u_{2n-1}$ and $u_n \dots u_{2n}$ for some n. The second words of δ and δ' are $u_{2n} \dots u_{4n-1}$ or $u_{2n} \dots u_{4n}$ and $u_{2n+1} \dots u_{4n+1}$ or $u_{2n+1} \dots u_{4n+2}$. In every case they overlap. Then, by induction, it can be shown that $w = (u_1 \dots u_{n-1})^{-1} w_0$ and then w is not ultimately periodic. Thus C is a weakly prefix code. □

Using the composition proposition 2.5 and the previous examples, it is easy to construct over a finite alphabet examples of codes having the same properties. Let $B = \{a, b\}$ and $\varphi : A \to B^+$ defined by: $\varphi(u_i) = a^i b$. The language $D = \varphi(A)$ is a prefix code.

Example 2.5 *Let C be the code defined in example 2.3. the language $C' = \varphi(C)$ is a weakly prefix code over a finite alphabet satisfying:*

- *every word has a countable (finite or not) number of C'-factorizations*
- *infinitely many words have a countable infinity of C'-factorizations.*

Example 2.6 *Let C be the code defined in example 2.4 the language $C' = \varphi(C)$ is a weakly prefix code over a finite alphabet and there exists a word: $\varphi(w_0)$ which has a noncountable infinity of C'-factorizations.*

3 The rational case

When a language C is rational, one can consider an automaton $\Omega_0 = (Q_0, q_0, q_F)$ with a finite set of states Q_0, a single initial state q_0 and a single final state q_F, which recognises C and such that no edge comes to q_0 and no edge goes from q_F. The automaton Ω_0 can be chosen trim (i.e. for every state q there exist a path from q_0 to q and a path from q to q_F) and unambiquous (i.e. the words of C have a single acceptance path). The automaton $\Omega = (Q, q_0, q_0)$ obtained by identification of q_0 and q_F recognises C^*. If C is a code, the automaton Ω is unambiguous [1]. This automaton looked as a Büchi automaton recognizes C^ω.

Theorem 3.1 *Let C be a rational language $\subset A^+$. The following conditions are equivalent:*

1. *C is a code*
2. *every infinite word has a finite number of C-factorizations*
3. *there exists p such that every infinite word has at most p C-factorizations.*

Proof. $3 \Rightarrow 2$: clear. $2 \Rightarrow 1$: This comes from theorem 1.7. $1 \Rightarrow 3$: Let C be a rational code and $\Omega = (Q, q_0, q_0)$ an unambiguous automaton for C^* constructed as said before. Consider $w \in C^\omega$ and $t \geq 1$. We call cut of (w,t) every sequence $(n_1, \ldots, n_{p-1})$ such that there exists n_p satisfying:

(i) $n_0 = 0 < n_1 < \ldots < n_{p-1} \leq t < n_p, p \geq 2$ and $w[n_{i-1}, n_i[\in C$ for $i = 1, \ldots, p$. (Here, and in the sequel, the factor $w_i w_{i+1} \ldots w_{j-1}$ of a word w is denoted by $w[i,j[$).

At first, we show that, for every t, (w,t) has at most Card(Q) cuts.

Let us consider $(n_1, \ldots, n_p)$ and $(n'_1, \ldots, n'_k)$ such that (i) is satisfied. Denote by q (resp. q') the state reached after reading $w[0,t[$ in the single successful path of $w[0,n_p[$ (resp. $w[0,n'_k[$). If $q = q', w[0,n_p[$ has a second successful path: path related to $w[0,n'_k[$ until t, path related to $w[0,n_p[$ after. Then $p = k$ and $(n_1, \ldots, n_{p-1}) = (n'_1, \ldots, n'_{p-1})$ since Ω is unambiguous.

Thus (w,t) has at most Card(Q) cuts. Then w has at most Card(Q) C-factorisations. □

Remark: An infinite word which has several C-factorizations is not necessarily ultimately periodic: the word: $ab^2cb^3(c^2b^3)cb^3(c^2b^3)^2 \ldots cb^3(c^2b^3)^n cb^3(c^2b^3)^{n+1} \ldots$ has two C-factorizations when $C = \{a, ab, bcb^2, bc^2b^2, b^2cb, b^2c^2b\}$.

A set of infinite words over an alphabet A is said to be rational if it is a finite union of sets $R_i S_i^\omega$ where R_i and S_i are rational subsets of A^*. It was proved that the rational sets of infinite words are the languages which can be recognized by a finite Büchi-automaton [4]. The set of rational subsets of A^ω is closed by finite union, finite intersection and complement [4]. For details, one can see [18].

Proposition 3.2 *Let C be a rational language $\subset A^+$. The set of infinite words which have several C-factorizations is rational.*

Proof. If C is rational, the semi-congruence defined by:

$$u \simeq v \Leftrightarrow u^{-1}C = v^{-1}C$$

is of finite index. Let us denote by $[u]$ the class of a word u. The set D of infinite words which have several C-factorizations can be written:

$$D = \bigcup_{[u] \subset C} C^*.[u].(C^\omega \cap ([u]^{-1}C - \{\varepsilon\})C^\omega).$$

So D is rational. □

Remarks:

– The set of infinite words which have several C-factorizations is countable when the code C has three elements [10]. It can be noncountable when the code C has

more that three elements. For example, let $C = \{ab, aba, bab^2, b^2ab^2a\}$. Every word of $aba(b^2ab^2a + bab^2ab)^\omega$ has a noncountable infinity of C-factorisations.

– It can be proved from proposition 3.2 that, if C is a rational language, C is an ω-code if and only if all its finite subsets are ω-codes. This property does not hold for nonrational languages as it can be seen for $C = \{ab\} \cup \{ab^nab^{n+1} | n > 0\}$.

– From proposition 3.2, we obtain the next statement which is a result of Staiger [20]. This statement agrees with the fact that a rational ω-language is specified by the set of ultimately periodic words contained in it [4].

Corollary 3.3 *Any rational weakly prefix code is an ω-code.*

Since it can be checked whether the rational set of infinite words which have several C-factorisations is empty or contains a periodic word, we have the following corollary.

Corollary 3.4 *One can decide whether a rational language is a π-code (resp. a weakly prefix code, or equivalently an ω-code).*

The membership problem for the studied classes of codes is decidable in the rational case. Indeed the result is well known for codes [1], and has been proved for codes with bounded deciphering delay by Cori [6]. This latter result is also a consequence of the next result of Capocelli, and can be also deduced from proposition 3.7.

Capocelli [5] gave a necessary and sufficient condition for a rational weakly prefix code (or ω-code) C to have a bounded deciphering delay. That is:

$$\exists p \geq 0 \forall u \in A^* uC^p A^* \cap C \neq \emptyset \Rightarrow C^+u \cap C^+ = \emptyset.$$

We give here another condition which obviously is satisfied when the code is finite. In this condition we need the notion of *adherence* [3]. An infinite word w belongs to $Adh(C)$, the adherence of a language C of finite words, if every left factor of w is a left factor of a word of C.

Lemma 3.5 *Let us consider a language $C \subset A^+$.*

1. *if C is a code having a finite deciphering delay, C is an ω-code and $C^\omega \cap C^*.Adh(C) = \emptyset$.*

2. *if C is a rational ω-code such that $C^\omega \cap C^*.Adh(C) = \emptyset$, then C has a bounded deciphering delay.*

Proof.

1. If C is not an ω-code C cannot be a code having finite deciphering delay (proposition 1.9). Thus, let C be an ω-code for which there is some $w \in C^\omega \cap C^*.\mathrm{Adh}(C)$. Without loss of generality, we may assume that $w = u_1u_2u_3 \ldots = u'_1u'_2 \ldots u'_pw'$ where $u_i, u'_i \in C$ for every $i, w' \in \mathrm{Adh}(C)$ and $u'_1 \neq u_1$ or $p = 0$. Since $w' \in \mathrm{Adh}(C)$, for every $d \geq 1$ there exists $v \in C$ such that $u_1 \ldots u_d \leq u'_1u'_2 \ldots u'_pv$ where $u'_1 \neq u_1$ or $p = 0$. Thus C has not the deciphering delay $(d-1)$.

2. Let C be a rational language and $\Omega = (Q, q_0, q_0)$ an unambiguous automaton for C^* constructed as said before. Let d be the number of states. Assume that C has not he delay d. There exist $n \geq 0, u_0, \ldots, u_d, u'_0, \ldots, u'_n \in C, z \in A^*$ such that $u_0 \ldots u_dz = u'_0 \ldots u'_n$ and $u'_0 \neq u_0$.

There exists a path of label $u'_0 \dots u'_n$ from q_0 to q_0. Within this path, we denote by q_j the state reached after reading $u_0 u_1 \dots u_j$. There exist j and $j' > j$ such that $q_j = q_{j'}$ (we denote $q = q_j$). Then we denote: $y = u_0 \dots u_j = u'_0 \dots u'_{m-1} x'$ with $x' < u'_m, x = u_{j+1} \dots u'_{j'}, x' x x^n = u'_m \dots u'_{m+h}$, with x^n suffix of $u'_{m+h}, u'_{j+1} \dots u_d z = x^n u'_{m+h+1} \dots u'_n$. If $h = 0$, for every $n, x' x^n x^n \in C$ and then $y x^\omega \in C^*.\mathrm{Adh}(C) \cap C^\omega$. If $h \geq 1, y x^\omega$ has two distinct C-factorisations: $u_0, \dots, u_j (u_{j+1}, \dots, u_{j'})^\omega$ and $u'_0, \dots, u'_{m-1}, (u'_m, \dots, u'_{m+h-1}, v)^\omega$ where $v = (u'_{m+h} x^{n-1}).(x^{-1} u'_m)$, thus C is not an ω-code. □

Lemma 3.5 can be used to derive a new proof of a result in [20]. To this end, we consider A^ω as a topological space defined by the set of open subsets: $E \subset A^\omega$ is open iff $E = WA^\omega$ for some $W \subset A^*$. The closed subsets (i.e. the complements of open subsets) are the languages of the form $\mathrm{Adh}(W)$ for some $W \subset A^*$ [21]. We need here the next classes of the Borel hierarchy. A F_σ-set is a countable union of closed subsets and a G_δ-set is a countable intersection of open subsets.

Corollary 3.6 *When C is a code with a finite deciphering delay, the language C^ω is a G_δ-set.*

Proof. Since $\mathrm{Adh}(C^*) = C^\omega \cup C^*.\mathrm{Adh}(C)$ [13], when $C^\omega \cap C^*.\mathrm{Adh}(C) = \emptyset$ the set C^ω is the difference of the closed set: $\mathrm{Adh}(C^*)$ and the F_σ-set: $C^*.\mathrm{Adh}(C)$, hence C^ω is a G_δ-set. □

Remark: The tempting assumption "$C^\omega = \cap C^i A^{\omega}$" is true for the codes C having a bounded deciphering delay [20] but no longer true for the codes C having a finite (but not bounded) deciphering delay (cf. example 3 of [20]).

We can summarize:

Theorem 3.7 *Let C be a rational language $\subset A^+$. The following conditions are equivalent:*

- *C is a code with a bounded deciphering delay*
- *C is a code with a finite deciphering delay*
- *C is an ω-code satisfying $C^\omega \cap C^*.Adh(C) = \emptyset$.*
- *C is a weakly prefix code satisfying $C^\omega \cap C^*.Adh(C) = \emptyset$.*

We have already seen that there exist ω-codes without finite deciphering delay. The other condition: "$C^*.\mathrm{Adh}(C) \cap C^\omega = \emptyset$" is neither sufficient. For example, the finite code $\{a, ab, bb\}$ is not an ω-code. Unfortunately proposition 3.7 is false when C is not rational. For example, let $C = \{ab^n c^n d | n \geq 0\} \cup \{a\} \cup b^* c$. Since $\mathrm{Adh}(C) = b^\omega \cup ab^\omega$, the ω-code C satisfies $C^\omega \cap C^*.\mathrm{Adh}(C) = \emptyset$, but the word a has no finite deciphering delay.

In the aim to be complete, let us now observe the finite case. The finite case is almost similar to the rational case. However proposition 3.7, as the result of Levenshtejn [12] and Capocelli [5], show that, in the finite case, the notion of ω-code and the notion of code with bounded deciphering delay coincide. This fact is also a result of Blanchard [2] which uses another notion of factorization ("découpage").

Nevertheless there are a lot of modifications when one considers two-element codes. Indeed, if $\{u, v\}$ is a code, $\{u, v\}$ is also an ω-code [10]. Since the examples given in this paper are chosen with three elements when it is possible, the obtained or recalled results can be recapitulated in the following proposition where $\mathbb{A}_{\mathrm{R}}$ (resp. $\mathbb{A}_{\mathrm{F}}, \mathbb{A}_2, \mathbb{A}_3$) denotes the class of rational (resp. finite, two-element, three-element) languages belonging to a given class of languages $\mathbb{A}$.

Proposition 3.8 *One has the following strict inclusions and equalities:*

$$\mathbf{B} \subset \mathbf{D} \subset \mathbf{I} \subset \mathbf{W} \subset \Pi \subset \mathbf{C}$$

moreover $\mathbf{B_R} = \mathbf{D_R}$ *and* $\mathbf{I_R} = \mathbf{W_R}$ *for rational sets,* $\mathbf{B_F} = \mathbf{D_F} = \mathbf{I_F} = \mathbf{W_F}$ *for finite sets and* $\mathbf{B_3} = \mathbf{D_3} = \mathbf{I_3} = \mathbf{W_3}$ *for three element sets, and finally* $\mathbf{B_2} = \mathbf{D_2} = \mathbf{I_2} = \mathbf{W_2} = \Pi_2 = \mathbf{C_2}$ *for two element sets.*

References

[1] J. Berstel and D. Perrin, *Theory of codes* (Academic Press, Orlando, 1985).

[2] F. Blanchard, Codes engendrant certains systèmes sofiques, *Theoret. Comput. Sci.* **68** (1989) 253-265.

[3] L. Boasson and M. Nivat, Adherences of languages, *J. Comput. System. Sci.*, **20**, (1980), 285-309.

[4] J.R. Buchi, On a decision method in restricted second-order arithmetic, *Proc. Congr. Logic,* Stanford Univ. Press, Stanford (1962) 1-11.

[5] R.M. Capocelli, Finite decipherability of weakly prefix codes, in *Algebra, Combinatorics and Logic in Computer Science* Coll. Math. Soc. J. Bolyai 42, North Holland, Amsterdam (1985) 175-184.

[6] R. Cori, Codes à délai borné maximaux, in *Théorie des codes* Publications du LITP, Universités de Paris VI et Paris VII (1979) 57-74.

[7] J. Devolder, Precircular codes and periodic biinfinite words, *Information and Computation,* **107** n° 2 (1993) 185-201.

[8] J. Devolder, Comportement des codes vis-à-vis des mots infinis et bi-infinis, in *Théorie des Automates et Applications,* (Edit. D. Krob, Rouen, 1991) 75-90.

[9] J. Devolder and E. Timmerman, Finitary codes for biinfinite words, *RAIRO Info. Theor. Appl.,* **26** n° 4 (1992) 363-386.

[10] J. Karhumäki, On three-element codes, *Theoret. Comput. Sci.* **40** (1985) 3-11.

[11] J.L. Lasses, Circular codes and synchronisation, *Internat. Journal of Computer and syst. Sci.* **5** (1976) 201-208.

[12] V.I. Levenshtejn, Some properties of coding and self adjusting automata for decoding messages, *Problemy Kibernetiki,* **11** (1964) 63-121.

[13] R. Lindner and L. Staiger, Algebraische Codierungstheorie, *Theorie der sequentiellen Codierungen,* Akademie-Verlag, Berlin (1977).

[14] I. Litovsky, Générateurs des langages rationnels de mots infinis, Thèse Univ. Lille I (1988).

[15] I. Litovsky and E. Timmerman, On generators of rational ω-power languages, *Theoret. Comput. Sci.* **53** (1987), 187-200.

[16] A. de Luca and A. Restivo, On some properties of very pure codes, *Theoret. Comput. Sci.,* (1980), 157-170.

[17] R. McNaughton, Testing and generating infinite sequences by a finite automaton, *Information and control,* **9** (1966), 521-530.

[18] D. Perrin et J.E. Pin, Mots infinis, *Publications du LITP,* Univ. Paris VI et VII, **91.06** (1991).

[19] M.P. Schutzenberger, Une théorie algébrique du codage, Séminaire Dubreil-Pisot **15** (1955-56), Institut Henri Poincaré, Paris.

[20] L. Staiger, On infinitary finite length codes, *RAIRO Theor. Inform. and Applic.* **20** (1986) n° 4, 483-494.

[21] L. Staiger and K. Wagner, Automatentheoretische und automatenfreie Charakterisierungen topologischer Klassen regulärer Folgenmengen, *Elektron. Inform.-Verarb. u. Kybernetik, EIK* **10** (1974), 379-392.

Received March 1, 1993

Acta Cybernetica, Vol. 11, No. 4, Szeged, 1994

A note on regular strongly shuffle-closed languages

B. Imreh * A. M. Ito†

In this work we study the class of regular strongly shuffle-closed languages and we present their description by giving a class of recognition automata.

The shuffle product operation plays an important role in the theory of formal languages, cf. [1], [2], [4]. Several properties of shuffle closed languages are studied in [3]. Among others a characterization of regular strongly shuffle-closed languages is presented by giving their expressions. Using this result, we determine a very simple class of deterministic automata accepting regular strongly shuffle-closed languages.

First of all we introduce some notions and notations. Let X be a nonempty finite set and let X^* denote the free monoid of words generated by X. We denote by 1 the empty word of X^*. The *shuffle product* of two words $u, v \in X^*$ is the set

$$u \diamond v = \{w : w = u_1 v_1 \dots u_k v_k, u = u_1 \dots u_k, v = v_1 \dots v_k, u_i, v_j \in X^*\}.$$

A language $L \subseteq X^*$ is called *shuffle-closed* if it is closed under $\diamond$, that is, if $u, v \in L$, then $u \diamond v \subseteq L$. If L is shuffle-closed and, for any $u \in L$, $v \in X^*$, the condition $u \diamond v \bigcap L \neq \emptyset$ implies $v \in L$, then L is called a *strongly shuffle-closed language*, or briefly, an *ssh-closed language*.

Next let $X = \{x_1, \dots, x_r\}$, $r \geq 1$, be an arbitrarily fixed alphabet. For any $L \subseteq X^*$, let us denote by $\mathrm{alph}(L)$ the set of elements of X occurring in words of L. We shall describe those regular ssh-closed languages over X for which $\mathrm{alph}(L) = X$.

We use the Parikh mapping and its inverse which are defined as follows. Let $N = \{0, 1, 2, \dots\}$. The mapping Ψ of X^* into the set N^r defined by

$$\Psi(u) = (\mu_{x_1}(u), \dots, \mu_{x_r}(u)), \quad u \in X^*,$$

is called the *Parikh mapping*, where $\mu_{x_t}(u)$ denotes the number of occurrences of x_t in u. For a language $L \subseteq X^*$, we define $\Psi(L) = \{\Psi(u) : u \in L\}$. Moreover, if $S \subseteq N^r$, then $\Psi^{-1}(S) = \{u : u \in X^* \ \& \ \Psi(u) \in S\}$.

Now we recall a notation and a result from [3].

Let $\mathbf{a} = (i_1, \dots, i_r)$, $\mathbf{b} = (j_1, \dots, j_r) \in N^r$ and let $p_1, \dots, p_r$ be positive integers. Then $\mathbf{a} \hookrightarrow \mathbf{b} (\mathrm{mod}\ (p_1, \dots, p_r))$ means that $i_t \geq j_t$ and $i_t \equiv j_t (\mathrm{mod}\ p_t)$, for all t, $t = 1, \dots, r$.

*Department of Informatics, A. József University, Árpád tér 2, H-6720 Szeged, Hungary
†Faculty of Science, Kyoto Sangyo University, 603 Kyoto, Japan

Theorem 1 *([3], Proposition 5.2) Let $L \subseteq X^*$ with $\mathrm{alph}(L) = X$. Then L is a regular ssh-closed language if and only if L is presented as*

$$L = \bigcup_{u \in F} \Psi^{-1}\Psi(u(x_1^{p_1})^* \dots (x_r^{p_r})^*)$$

where
(i) $p_1, \dots, p_r$ are positive integers,
(ii) F is a finite language over X with $1 \in F$ satisfying
(ii)-(1) for any $u \in F$, we have $0 \le j_t < p_t$, $1 \le t \le r$ where $\Psi(u) = (j_1, \dots, j_r)$,
(ii)-(2) for any $u, v \in F$, there is a $w \in F$ such that $\Psi(uv) \hookrightarrow \Psi(w) (\mathrm{mod}\ (p_1, \dots, p_r))$,
(ii)-(3) for any $u, v \in F$, there is a $w \in F$ such that $\Psi(uw) \hookrightarrow \Psi(v) (\mathrm{mod}\ (p_1, \dots, p_r))$.

Finally, we make some further preparation. For any positive integer p and $x_t \in X$, let us denote by $\mathrm{C}^{(p,x_t)} = (X, \{0, \dots, p-1\}, \delta^{(p,x_t)})$ the automaton defined by the following transition function. For any $j \in \{0, \dots, p-1\}$, $x \in X$, let

$$\delta^{(p,x_t)}(j, x) = \begin{cases} j & \text{if } x \neq x_t, \\ j + 1 (\mathrm{mod}\ p) & \text{if } x = x_t \end{cases}$$

where $j + 1 (\mathrm{mod}\ p)$ denotes the least nonnegative residue of $j + 1$ modulo p.

Now let $p_1, \dots, p_r$ be positive integers and form the direct product of the automata $\mathrm{C}^{(p_t,x_t)}$, $t = 1, \dots, r$. Let us denote by $\mathrm{C}^{(p_1,\dots,p_r)}$ this direct product and by $\delta^{(p_1,\dots,p_r)}$ its transition function. It is easy to prove that $\mathrm{C}^{(p_1,\dots,p_r)}$ has the following properties:

(a) it is a commutative automaton,

(b) if $\mathbf{a}, \mathbf{b} \in \prod_{t=1}^{r} \{0, \dots, p_t - 1\}$, $u \in X^*$ are such that $\delta^{(p_1,\dots,p_r)}(\mathbf{a}, u) = \mathbf{b}$, then $\delta^{(p_1,\dots,p_r)}(\mathbf{a}, v) = \mathbf{b}$, for all $v \in \Psi^{-1}\Psi(u)$,

(c) for any $u \in X^*$, $\delta^{(p_1,\dots,p_r)}(\mathbf{0}, u) = \Psi(u) (\mathrm{mod}\ (p_1, \dots, p_r))$,

where $\mathbf{0}$ denotes the r-dimensional 0-vector and $\Psi(u) (\mathrm{mod}\ (p_1, \dots, p_r))$ denotes the vector $(i_1 (\mathrm{mod}\ p_1), \dots, i_r (\mathrm{mod}\ p_r))$ with $\Psi(u) = (i_1, \dots, i_r)$.

For each t, $t = 1, \dots, r$, let us denote by $\mathcal{M}_{p_t}$ the group defined by the addition mod p_t over the set $\{0, \dots, p_t - 1\}$. Let $\mathcal{M}^{(p_1,\dots,p_r)}$ denote the direct product of the groups $\mathcal{M}_{p_t}$, $t = 1, \dots, r$. Then $\mathcal{M}^{(p_1,\dots,p_r)}$ is also a group; let $\oplus$ denote its operation. Let us observe that the set of states of $\mathrm{C}^{(p_1,\dots,p_r)}$ is equal to the set of elements of $\mathcal{M}^{(p_1,\dots,p_r)}$. Therefore, for any subgroup H of $\mathcal{M}^{(p_1,\dots,p_r)}$, we can define the recognizer

$$\mathbf{R}_H^{(p_1,\dots,p_r)} = (\prod_{t=1}^{r} \{0, \dots, p_t - 1\}, X, \delta^{(p_1,\dots,p_r)}, \mathbf{0}, H),$$

where $\mathbf{0}$ is the initial state and H is the set of the final states.

The next property of $\mathbf{R}_H^{(p_1,\dots,p_r)}$ can be proved easily:

(d) if $u, v \in X^*$ are accepted by $\mathbf{R}_H^{(p_1,\dots,p_r)}$ with final states $\mathbf{a}$, $\mathbf{b}$, respectively, then uv is also accepted by $\mathbf{R}_H^{(p_1,\dots,p_r)}$ with the final state $\mathbf{a} \oplus \mathbf{b}$.

Finally, form the set of recognizers

$$\mathcal{M}_X = \{\mathbf{R}_H^{(p_1,\ldots,p_r)} : (p_1,\ldots,p_r) \in N^r \text{ and } H \text{ is a subgroup of } \mathcal{M}^{(p_1,\ldots,p_r)}\}.$$

Now we are ready to prove our result.

Theorem 2 *A language $L \subseteq X^*$ with $\mathrm{alph}(L) = X$ is regular ssh-closed if and only if L is accepted by a recognizer from $\mathcal{M}_X$.*

Proof. In order to prove the necessity, let us suppose that $L \subseteq X^*$ is a regular ssh-closed language with $\mathrm{alph}(L) = X$. Then there are positive integers $p_1,\ldots,p_r$ and $F \subseteq X^*$ which satisfy the conditions of Theorem 1. Let us consider the automaton $\mathbf{C}^{(p_1,\ldots,p_r)}$ and let us define the set H by

$$H = \{\mathbf{a} : \mathbf{a} \in \prod_{t=1}^{r}\{0,\ldots,p_t-1\} \text{ and } \delta^{(p_1,\ldots,p_r)}(0,u) = \mathbf{a}, \text{ for some } u \in F\}.$$

We show that H is a subgroup of $\mathcal{M}^{(p_1,\ldots,p_r)}$. Indeed, let $\mathbf{a},\mathbf{b} \in H$ be arbitrary elements. By the definition of H, there are $u,v \in F$ with $\delta^{(p_1,\ldots,p_r)}(0,u) = \mathbf{a}$ and $\delta^{(p_1,\ldots,p_r)}(0,v) = \mathbf{b}$. Let $\Psi(u) = (i_1,\ldots,i_r)$ and $\Psi(v) = (j_1,\ldots,j_r)$. Then, by $(ii)-(1)$, we have $0 \le i_t, j_t < p_t$, for all $t = 1,\ldots,r$, and hence, we obtain, by (c), that $\mathbf{a} = (i_1,\ldots,i_r)$ and $\mathbf{b} = (j_1,\ldots,j_r)$. On the other hand, by (ii)-(2) of Theorem 1, there exists a $w \in F$ with $\Psi(uv) \hookrightarrow \Psi(w)(\mathrm{mod}\ (p_1,\ldots,p_r))$. Let $\Psi(w) = (k_1,\ldots,k_r)$. Then, by $(ii)-(1)$ and (c), $\delta^{(p_1,\ldots,p_r)}(0,w) = (k_1,\ldots,k_r)$. Since $w \in F$, we have $(k_1,\ldots,k_r) \in H$. From $\Psi(uv) \hookrightarrow \Psi(w)$ it follows that $i_t + j_t \equiv k_t(\mathrm{mod}\ p_t)$, $t = 1,\ldots,r$. But then $\mathbf{a} \oplus \mathbf{b} = (k_1,\ldots,k_r)$. Therefore, H is closed under the operation $\oplus$ implying that H is a subgroup of $\mathcal{M}^{(p_1,\ldots,p_r)}$. This completes the proof of the necessity.

In order to prove the sufficiency, let us suppose that $L \subseteq X^*$ with $\mathrm{alph}(L) = X$ and there exists a recognizer $\mathbf{R}_H^{(p_1,\ldots,p_r)} \in \mathcal{M}_X$ accepting L. We show that L is a regular ssh-closed language.

The regularity of L is obvious. Now let $u,v \in L$ and let w be an arbitrary element of the set $u \diamond v$. Since L is accepted by $\mathbf{R}_H^{(p_1,\ldots,p_r)}$, there are $\mathbf{a},\mathbf{b} \in H$ such that $\delta^{(p_1,\ldots,p_r)}(0,u) = \mathbf{a}$ and $\delta^{(p_1,\ldots,p_r)}(0,v) = \mathbf{b}$. Therefore, by (d), we obtain that uv is accepted by $\mathbf{R}_H^{(p_1,\ldots,p_r)}$ with the final state $\mathbf{a} \oplus \mathbf{b}$. From this, by (b), we get that $w \in L$, and so, L is shuffle-closed.

Finally, let $u \in L$, $v \in X^*$ and let us assume that $u \diamond v \bigcap L \neq \emptyset$. If $v = 1$, then $\delta^{(p_1,\ldots,p_r)}(0,v) = \mathbf{0} \in H$, and so, $v \in L$. Now let us suppose that $v \neq 1$. Let $\delta^{(p_1,\ldots,p_r)}(0,u) = \mathbf{a}$, $\delta^{(p_1,\ldots,p_r)}(0,v) = \mathbf{b}$ and let $\Psi(u) = (i'_1,\ldots,i'_r)$, $\Psi(v) = (j'_1,\ldots,j'_r)$. Then there exist nonnegative integers $i_t < p_t$, $j_t < p_t$, l_t, k_t, $t = 1,\ldots,r$, such that $i'_t = i_t + l_t p_t$, $j'_t = j_t + k_t p_t$, $t = 1,\ldots,r$. Let us denote by u' and v' the words $x_1^{i_1+l_1p_1}\ldots x_r^{i_r+l_rp_r}$ and $x_1^{j_1+k_1p_1}\ldots x_r^{j_r+k_rp_r}$, respectively. Using (b) and (c), we obtain that $\delta^{(p_1,\ldots,p_r)}(0,u') = \mathbf{a}$, $\delta^{(p_1,\ldots,p_r)}(0,v') = \mathbf{b}$, where $\mathbf{a} = (i_1,\ldots,i_r)$, $\mathbf{b} = (j_1,\ldots,j_r)$. By our assumption on $u \diamond v$, there exists a word $w \in u \diamond v \bigcap L$. Let

$$w' = x_1^{i_1+j_1+(l_1+k_1)p_1}\ldots x_r^{i_r+j_r+(l_r+k_r)p_r}.$$

Since $w \in u \diamond v \bigcap L$ and $\Psi(w') = \Psi(u'v') = \Psi(uv) = \Psi(w)$, (b) implies $w' \in L$. On the other hand, by (c), we have

$$\delta^{(p_1,\ldots,p_r)}(0, w') = (i_1 + j_1 (\mathrm{mod}\ p_1), \ldots, i_r + j_r (\mathrm{mod}\ p_r)).$$

Now let us observe that $(i_1 + j_1 (\mathrm{mod}\ p_1), \ldots, i_r + j_r (\mathrm{mod}\ p_r)) = \mathbf{a} \oplus \mathbf{b}$. Since $w' \in L$, we have $\mathbf{a} \oplus \mathbf{b} \in H$. But H is a subgroup of $\mathcal{M}^{(p_1,\ldots,p_r)}$, thus $\mathbf{a} \in H$ and $\mathbf{a} \oplus \mathbf{b} \in H$ imply $\mathbf{b} \in H$. Therefore, by $\delta^{(p_1,\ldots,p_r)}(0, v) = \mathbf{b}$, we obtain that $v \in L$, and so, L is an ssh-closed language. This completes the proof of the theorem.

References

[1] Eilenberg, S., *Automata, Languages and Machines,* Vol A, Academic Press, New York, 1974.

[2] Ginsburg, S., *Algebraic and Automata-Theoretic Properties of Formal Languages*, North-Holland Publ., Amsterdam, 1975.

[3] Ito, M., Thierrin, G., Yu, S.S., Shuffle-closed Languages, Publicationes Mathematicae, submitted for publication.

[4] Lothaire, M., *Combinatorics on Words*, Encyclopedia of Mathematics and its appli-cations, Addition-Wesley, Reading, 1983.

Received September 1, 1994

Acta Cybernetica, Vol. 11, No. 4, Szeged, 1994

A Pumping Lemma for Output Languages of Attributed Tree Transducers

A. Kühnemann* H. Vogler*

Abstract

An attributed tree transducer is a formal model for studying properties of attribute grammars. In this paper we introduce and prove a pumping lemma for output languages of noncircular, producing, and visiting attributed tree transducers. We apply this pumping lemma to gain two results: (1) there is no noncircular, producing, and visiting attributed tree transducer which computes the set of all monadic trees with exponential height as output and (2) there is a hierarchy of noncircular, producing, and visiting attributed tree transducers with respect to their number of attributes.

1 Introduction

In formal language theory we are often confronted with the task to decide, whether a given language L is an element of a class $\mathcal{L}$ of languages, where $\mathcal{L}$ usually is defined by a class of grammars or translation schemes. If L is an element of $\mathcal{L}$, then we have to specify a grammar or a translation scheme which generates L. If L is not an element of $\mathcal{L}$, then sometimes we can use necessary conditions which every language in $\mathcal{L}$ has to fulfill. With the help of these conditions we can try to deduce a contradiction to the assumption that L is an element of $\mathcal{L}$. Pumping lemmata are such necessary conditions which have been proven to be very useful tools.

Pumping lemmata have been invented for different kinds of languages, for example string languages, graph and hypergraph languages, picture languages, and tree transducer languages.

*Institut für Softwaretechnik I, Fakultät Informatik, Technische Universität Dresden, D–01062 Dresden, Germany, e–mail: {ak15, hv3}@irs.inf.tu-dresden.de

In the case of string languages we can observe the following evolution of pumping lemmata: Scheinberg has used in [Sch60] a proof technique which can be seen as a predecessor of the well known pumping lemma for context–free languages of Bar–Hillel, Perles, and Shamir [BPS61]. The structure of the latter pumping lemma has served as pattern for most of the existing pumping lemmata in the literature and therefore it seems to be the root of the research about pumping lemmata. Since it also has influenced our pumping lemma, we present here a short version of the lemma's central statement and we recall its proof idea:

For every context–free grammar G there is a natural number n_G, called the pumping index of G, such that for every string z which is an element of the language $L(G)$ generated by G and which has at least the length n_G, the following holds. There is a decomposition $z = uvwxy$, such that v or x is not the empty string and such that for every natural number j, the pumped string uv^jwx^jy is an element of $L(G)$.

The proof can be sketched as follows: We choose a sufficiently long string z of $L(G)$, such that its derivation tree e has the following property: e is high enough, such that it has a path p, on which two different nodes x_1 and x_2 are labeled by the same nonterminal symbol. Assuming that x_1 is closer to the root of e than x_2, we can define the following tree $\bar{e}$: Roughly speaking, the tree $\bar{e}$ is that part of e which has x_1 as root and from which the subtree rooting at x_2 is pruned. Since x_1 and x_2 have the same label, we can construct for every natural number j a new derivation tree, by repeating $\bar{e}$ j times. Taking the yield of these derivation trees, we obtain new elements of $L(G)$.

As stated above, the pumping lemma of Bar–Hillel, Perles, and Shamir is only a necessary condition for the context–freeness of a string language. Thus there exist non–context–free languages which fulfill the requirements of the pumping lemma. In the sequel more and more stronger pumping lemmata for context–free string languages have been invented. Most of them, however, represent no sufficient condition for context–freeness. For example, in the Ogden–Lemma (cf. [Ogd68]) we can designate distinguished positions in the pumped string. This allows us to concentrate on those substrings, in which pumping is effective. Bader and Moura have developed in [BM82] a stronger version, the Generalized Ogden–Lemma, where additionally positions in the pumped string can be excluded. In the paper of Bader and Moura it is also shown that there is no stronger version of the Generalized Ogden–Lemma which exactly characterises the context–free string languages.

Wise has introduced in [Wis76] his Strong Pumping Lemma which is a necessary and sufficient condition for context–free string languages. The central idea of this lemma is to pump sentential forms of a grammar for a context–free language L instead of pumping terminal strings of L. The Strong Pumping Lemma of Wise represents another method to prove that a certain language is not context–free by assuming that it is context–free and by applying the lemma. In contrast to the other pumping lemmata stated above, this application guarantees the existence of a contradiction, because the Strong Pumping Lemma characterises the class of context–free languages. Clearly, it depends on the skill of the researcher, whether he can construct this contradiction, yes or no.

There also exist pumping lemmata for subclasses of the class of context–free languages: Boonyavatana and Slutzki have invented pumping lemmata for linear context- -free and nonterminal bounded string languages in [BS86a] and [BS86b], respectively. Yu has developed in [Yu89] a pumping lemma for deterministic context–free languages. Ehrenfeucht, Parikh, and Rozenberg have introduced in [EPR81] the Block Pumping Lemma as characterisation of regular string languages.

There are also pumping lemmata in the area of context– free graph and hypergraph languages: Kreowski (cf. [Kre79]) and Habel (cf. [Hab89]) have invented pumping lemmata for edge–replacement and hyperedge–replacement languages, respectively. These pumping lemmata require a certain size of the pumped graphs. In comparison with them, the Maximum Path Length Pumping Lemma for edge–replacement languages of Kuske (cf. [Kus91,Kus93]) needs a certain length of a path in the pumped graphs.

Another kind of language paradigm are the picture languages. Hinz has developed in [Hin90] pumping lemmata for certain subclasses of picture languages.

First Aho and Ullman have inspected pumping lemmata for output languages of translation schemes in [AU71], namely for generalized syntax directed translations. Perrault and Ésik have introduced in [Per76] and [Ési80], respectively, pumping lemmata for (nondeterministic) top–down tree transducers (cf. [Rou70, Tha70,Eng75]). The results of Ésik also appear in the book of Gécseg and Steinby (cf. [GS83]). Engelfriet, Rozenberg, and Slutzki have presented in [ERS80] a pumping lemma for deterministic top–down tree–to–string transducers which has a structure that is closely related to the pumping lemma for context–free string languages. The proof of this lemma had a big influence on the development of the pumping lemma for attributed tree transducers which we present in this paper.

The concept of attributed tree transducer has been invented by Fülöp in [Fül81]; it is a formal model for studying properties of attribute grammars introduced by Knuth in [Knu68]. Attributed tree transducers are abstractions of attribute grammars in the sense that they take trees over an arbitrary ranked alphabet of input symbols rather than derivation trees as argument, and that the values of the attributes are also trees over a ranked alphabet of output symbols.

Like in attribute grammars, the set of attributes is partitioned into the set of synthesized and inherited attributes which are associated to the input symbols and which compute their values in a bottom–up manner and in a top–down manner, respectively. In contrast to attribute grammars, to every input symbol the whole set of attributes is associated; this means that all attributes are available at any node of any input tree. Roughly speaking, computing the value of a synthesized attribute occurrence of a node x of an input tree, the values of the inherited attribute occurrences of x and of the synthesized attribute occurrences of its sons (if they exist) may be used and, computing the value of an inherited attribute occurrence of x, the values of the inherited attribute occurrences of its father (if it exists) and of the synthesized attribute occurrences of x and of its brothers may be used. This refers to the usual Bochmann Normal Form of attribute grammars [Boc76].

In this paper we consider only total deterministic attributed tree transducers: For every node x of an input tree which is labeled by a particular input symbol

and for every synthesized attribute s, the computation of the attribute occurrence of s at x is fixed by exactly one rule. Similarly, for every node x which is labeled by a particular input symbol and for every inherited attribute i, the computation of the attribute occurrence of i at the j–th son of x is fixed by exactly one rule.

As in attribute grammars, these dependencies can induce circularities among the attribute occurrences of an input tree. We restrict the attributed tree transducers to be noncircular and we designate a synthesized attribute as initial attribute. Thus we designate an initial attribute occurrence at the root of every input tree of which the value will be the output tree. Then every attributed tree transducer M computes a total function from input trees to output trees. This function is called the tree transformation of M. The output language of an attributed tree transducer M is defined as the range of the tree transformation of M.

As stated at the beginning of the introduction, pumping lemmata can help us to prove that a certain language is not an element of a class of languages. But not only pumping lemmata have been used to solve such a kind of problem: Fülöp and Vágvölgyi have shown in [FV91] by means of a direct proof that a particular tree transformation (which is induced by a bottom–up tree transducer; cf. [Eng75]) cannot be computed by an attributed tree transducer. Maybe the proof of Fülöp and Vágvölgyi can be generalized to a proof of a kind of pumping lemma. But we do not follow here this line of generalization and return to the development of a pumping lemma for a particular class of attributed tree transducers.

We restrict our pumping lemma to special attributed tree transducers, namely producing and visiting (and noncircular) attributed tree transducers. An attributed tree transducer is producing, if every rule application delivers at least one new output symbol. An attributed tree transducer is visiting, if for every input tree and for every node x of it, the value of at least one attribute occurrence of x is needed to compute the value of the initial synthesized attribute occurrence at the root.

The main idea of our pumping lemma for output languages of producing and visiting attributed tree transducers is adopted from the proof of the pumping lemma for context–free string languages that was outlined at the beginning of this introduction. In the case of context–free string languages we have to inspect a derivation tree of a sufficiently long string to deduce new pumped strings. Here we have to consider input trees belonging to a sufficiently large output tree to obtain new pumped output trees: For every producing and visiting attributed tree transducer M, a natural number n_M, called the pumping index of M, can be constructed. If we choose an output tree t from the output language of M which has at least n_M nodes, then every input tree e which can be transformed into t has the following property: e is high enough, such that it has a path p, on which two different nodes x_1 and x_2 can be found, which have the same set of attribute occurrences that are needed to calculate the initial attribute occurrence at the root of e. Assuming that x_1 is closer to the root of e than x_2, we can define the following tree $\bar{e}$: Roughly speaking, the tree $\bar{e}$ is that part of e which has x_1 as root and from which the subtree rooting at x_2 is pruned. Since the two nodes are compatible with respect to the needed attribute occurrences, we can construct new input trees by repeat-

ing $\bar{e}$ arbitrarily many times. Translating these input trees by M, we obtain new elements of the output language of M.

The proof is based on the observation that the decomposition of the input tree e induces a decomposition of the output tree t into output patterns and that these patterns are used to construct the new output trees. Thus the pumping process itself can be described by using only the output patterns. Therefore the applications of the pumping lemma are completely independend of the underlying input trees.

In this paper we apply our pumping lemma to prove the following two results:

- There is no noncircular, producing, and visiting attributed tree transducer which computes the set of all monadic trees with exponential height as output.
- There is a hierarchy of noncircular, producing, and visiting attributed tree transducers with respect to their number of attributes.

This paper is divided into five sections, from which this one is the first. In Section 2 we fix all the notions and notations, especially about attributed tree transducers, which are necessary for the remaining sections. Section 3 contains the pumping lemma together with its proof. In Section 4 we show the two applications of the pumping lemma. Finally, in Section 5 the reader can find a short summary and a presentation of further research topics.

2 Preliminaries

In this section we collect the notations, notions, and definitions which are used throughout this paper. Most of the definitions are taken from [KV94], some of them with a slight modification.

2.1 General notations

We denote the set of natural numbers (including 0) by $\mathbb{N}$. For every $m \in \mathbb{N}$, the set $\{1, \ldots, m\}$ is denoted by $[m]$, thus $[0]$ denotes the empty set $\emptyset$. The empty word is denoted by ε. For an arbitrary set S, the cardinality of S is denoted by $card(S)$ and the set of all subsets of S is denoted by $\mathcal{P}(S)$. If S is a subset of $\mathbb{N}$, then $max(S)$ denotes the maximum of S; $max(\emptyset)$ is defined as 0. A relation $f \subseteq A \times B$ is a *partial function*, if for every $(a, b_1) \in f$ and $(a, b_2) \in f$, the elements b_1 and b_2 are equal. Such a partial function is denoted by $f : A \dashrightarrow B$.

If A is an alphabet, then A^* denotes the set of words over A. For a string v and two lists $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ of strings such that no pair u_i and u_j overlaps in v, we abbreviate by $v[u_1/v_1, \ldots, u_n/v_n]$ the string which is obtained from v by replacing every occurrence of u_i in v by v_i. The resulting string is also denoted by $v[u_i/v_i \,;\, i \in [n]]$. $|p|$ denotes the length of a string p over an alphabet which should be known from the context. If P_1 and P_2 are two sets of strings, then $P_1 \cdot P_2 := \{p_1 p_2 \mid p_1 \in P_1, p_2 \in P_2\}$.

Let $\Rightarrow$ be a binary relation on some set T. Then, $\Rightarrow^*$ and $\Rightarrow^+$ denote the transitive, reflexive closure of $\Rightarrow$ and the transitive closure of $\Rightarrow$, respectively. Let $n \in \mathbb{N} - \{0\}$. If $t_j \in T$ for every $j \in [n+1]$ and if $t_j \Rightarrow t_{j+1}$ for every $j \in [n]$, then the sequence $t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_{n+1}$ is called a *derivation*. If only the first element t_1 and the last element t_{n+1} of a derivation are important, we also use the notation $t_1 \Rightarrow^+ t_{n+1}$. Note that there can exist more than one derivation $t_1 \Rightarrow^+ t_{n+1}$. If $t \Rightarrow^* t'$ for $t, t' \in T$ and if there is no $t'' \in T$ such that $t' \Rightarrow t''$, then t' is called a *normal form of t with respect to* $\Rightarrow$. In general t can have either no or one or more than one normal form. If the normal form of t exists and if it is unique, then it is denoted by $nf(\Rightarrow, t)$. The relation $\Rightarrow$ is *confluent*, if for every $t, t_1, t_2 \in T$ with $t \Rightarrow^* t_1$ and $t \Rightarrow^* t_2$, there is an $t' \in T$ such that $t_1 \Rightarrow^* t'$ and $t_2 \Rightarrow^* t'$. It is *noetherian* or *terminating*, if there is no infinite derivation of $\Rightarrow$. If $\Rightarrow$ is noetherian and confluent, then for every $t \in T$, the normal form of t exists and it is unique.

2.2 Ranked alphabets, trees, and tree transformations

A *ranked alphabet* is a pair $(\Sigma, rank_\Sigma)$ where Σ is a finite set and $rank_\Sigma : \Sigma \longrightarrow \mathbb{N}$ is a mapping which associates with every symbol a natural number called the rank of the symbol. If $\sigma \in \Sigma$ with $rank_\Sigma(\sigma) = n$, and Σ is clear from the context, then we also write $\sigma^{(n)}$ and $rank(\sigma) = n$. If the rank function is clear from the context, then it is dropped from the notation. The set of elements with rank n is denoted by $\Sigma^{(n)}$.

For a ranked alphabet Σ, the set of *trees over* Σ, denoted by $T\langle\Sigma\rangle$, is the smallest subset $T \subseteq (\Sigma \cup \{(,),,\})^*$ such that for every $\sigma \in \Sigma^{(n)}$ with $n \geq 0$ and $t_1, \ldots, t_n \in T$, the string $\sigma(t_1, \ldots, t_n) \in T$. For a symbol $\sigma \in \Sigma^{(0)}$ we simply write σ instead of $\sigma()$.

The following functions are defined inductively on the structure of trees in $T\langle\Sigma\rangle$ (here, the induction base is a special case of the induction step):

- $height : T\langle\Sigma\rangle \longrightarrow \mathbb{N}$ delivers the *height of a tree* $t \in T\langle\Sigma\rangle$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $height(\sigma(t_1, \ldots, t_n)) = 1 + max(\{height(t_i) \mid i \in [n]\})$.

- $size_{\Sigma'} : T\langle\Sigma\rangle \longrightarrow \mathbb{N}$ delivers the *size of a tree* $t \in T\langle\Sigma\rangle$ *with respect to a subset* $\Sigma' \subseteq \Sigma$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $size_{\Sigma'}(\sigma(t_1, \ldots, t_n)) = 1 + \sum_{i \in [n]} size_{\Sigma'}(t_i)$, if $\sigma \in \Sigma'$,
  $size_{\Sigma'}(\sigma(t_1, \ldots, t_n)) = \sum_{i \in [n]} size_{\Sigma'}(t_i)$, if $\sigma \notin \Sigma'$.
  If $\Sigma' = \Sigma$, then we abbreviate $size_{\Sigma'}$ by $size$.

- $paths : T\langle\Sigma\rangle \longrightarrow \mathcal{P}(\mathbb{N}^*)$ delivers the *set of paths of a tree* $t \in T\langle\Sigma\rangle$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $paths(\sigma(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \{p \mid p = ip', i \in [n], p' \in paths(t_i)\}$.

- *label* : $T\langle\Sigma\rangle \times \mathbb{N}^* - \rightarrow \Sigma$ delivers the *label of the node of a tree* $t \in T\langle\Sigma\rangle$ *reached by a path* $p \in paths(t)$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $label(\sigma(t_1, \ldots, t_n), p) = \sigma$, if $p = \varepsilon$,
  $label(\sigma(t_1, \ldots, t_n), p) = label(t_i, p')$, if $p = ip'$ for some $i \in [n]$.

- *subtree* : $T\langle\Sigma\rangle \times \mathbb{N}^* - \rightarrow T\langle\Sigma\rangle$ delivers the *subtree of a tree* $t \in T\langle\Sigma\rangle$ *reached by a path* $p \in paths(t)$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $subtree(\sigma(t_1, \ldots, t_n), p) = \sigma(t_1, \ldots, t_n)$, if $p = \varepsilon$,
  $subtree(\sigma(t_1, \ldots, t_n), p) = subtree(t_i, p')$, if $p = ip'$ for some $i \in [n]$.

- *repl* : $T\langle\Sigma\rangle \times \mathbb{N}^* \times T\langle\Sigma\rangle - \rightarrow T\langle\Sigma\rangle$ delivers the *tree obtained from a tree* $t \in T\langle\Sigma\rangle$ *by replacing the subtree reached by a path* $p \in paths(t)$, *by another tree* $t' \in T\langle\Sigma\rangle$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $repl(\sigma(t_1, \ldots, t_n), p, t') = t'$, if $p = \varepsilon$,
  $repl(\sigma(t_1, \ldots, t_n), p, t') = \sigma(t_1, \ldots, repl(t_i, p', t'), \ldots, t_n)$, if $p = ip'$ for some $i \in [n]$.
  In the following we use the more convenient notation $t[p \leftarrow t']$ instead of $repl(t, p, t')$.

For every tree $t \in T\langle\Sigma\rangle$ and for every path $p \in paths(t)$, the path p determines exactly one node of t. This node will be denoted by $node(t, p)$.

Let Σ be a ranked alphabet, $t \in T\langle\Sigma\rangle$, and let U be another ranked alphabet with $rank(u) = 0$ for every $u \in U$ and with $U \cap \Sigma = \emptyset$. A tree $t' \in T\langle\Sigma \cup U\rangle$ is called a *pattern in* $t \in T\langle\Sigma\rangle$, if there is a symbol $v \notin \Sigma$ with $rank(v) = 0$, there is a tree $t'' \in T\langle\Sigma \cup \{v\}\rangle$, and for every $u \in U$ there is a tree $t_u \in T\langle\Sigma\rangle$, such that $t = t''[v/t'[u/t_u \ ; \ u \in U]]$.

A *tree transformation* is a total function $\tau : T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ where Σ and Δ are ranked alphabets.

2.3 Attributed Tree Transducers

In this subsection we define the syntax of so called *si*–tree transducers and the derivation relations which are induced by them. In [Gie88] *si*–tree transducers are called full attributed tree transducers. Though *si*–tree transducers are an extension of attributed tree transducers in the sense of [Fül81], we also use simply the notion attributed tree transducer for an *si*–tree transducer. If we restrict the transducers to be noncircular, then their derivation relations are confluent and noetherian, and every noncircular transducer computes a tree transformation.

A system of attributes is the first component in the definition of an attributed tree transducer M. We specify a ranked input alphabet Σ. Then, intuitively, M takes an argument e where e is a tree over Σ, called *input tree*, on which the evaluation of attribute values is performed. An *output tree* is built up over a ranked alphabet Δ of working symbols. The derivations of M will start with an initial

synthesized attribute s_{in} and with an extra marker *root* on top of the input tree where *root* is a new symbol of rank 1. If e is an input tree, then in anology to [KV94] we call the tree $\tilde{e} = root(e)$ the *control tree*, because it controls the derivation of the transducer (cf. Figure 1). The role of the marker *root* is explained after defining the derivation relation. Of course, the kernel of the definition of an attributed tree transducer is the finite set of rewrite rules. The possible right-hand sides of rules are fixed at the end of the definition.

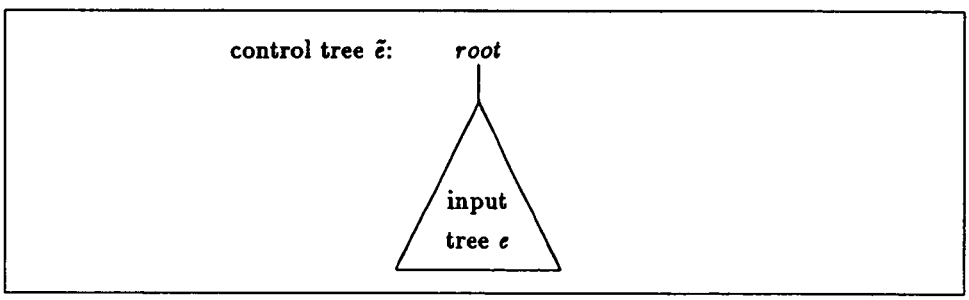


Figure 1: The input tree e and the control tree $\tilde{e}$.

We mention already here that, similarly to top–down tree transducers, we designate the argument position of every attribute to contain the control tree $\tilde{e}$. Additionally, in attributed tree transducers the control tree $\tilde{e}$ is associated with a path through $\tilde{e}$. Actually, in the argument of an attribute, only a path through $\tilde{e}$ will occur, the control tree itself will parameterize the derivation relation (cf. Definition 2.6).

Definition 2.1 An *si-tree transducer* is a tuple $(\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ where

- $\mathcal{A} = (A, A_s, A_i)$ is a system of attributes, where
  - A is a ranked alphabet of *attributes*; for every $a \in A$, $rank_A(a) = 1$.
  - $A_s \subseteq A$ and $A_i \subseteq A$ are the disjoint sets of *synthesized attributes* and *inherited attributes*, respectively, with $A = A_s \cup A_i$.
- Δ is the ranked alphabet of *working symbols* (or: *output symbols*) with $A \cap \Delta = \emptyset$.
- Σ is the ranked alphabet of *input symbols* with $A \cap \Sigma = \emptyset$.
- $s_{in} \in A_s$ is the *initial attribute*.
- *root* is a symbol of rank 1, called the *root marker*, where $root \notin A \cup \Delta \cup \Sigma$.

- $R = \bigcup_{\sigma \in \Sigma \cup \{root\}} R_\sigma$ is a finite set of *rules*, defined by Conditions 1. and 2.

1. The set R_{root} contains exactly one rule of the form

$$s_{in}(z) \to \rho$$

with $\rho \in RHS(A_s, \emptyset, \Delta, root)$.
For every $i \in A_i$, the set R_{root} contains exactly one rule of the form

$$i(z1) \to \rho$$

with $\rho \in RHS(A_s, \emptyset, \Delta, root)$.

2. For every $\sigma \in \Sigma^{(k)}$ with $k \geq 0$ and for every $s \in A_s$, the set R_σ contains exactly one rule of the form

$$s(z) \to \rho$$

with $\rho \in RHS(A_s, A_i, \Delta, \sigma)$.
For every $\sigma \in \Sigma^{(k)}$ with $k \geq 0$, for every $i \in A_i$ and for every $j \in [k]$, the set R_σ contains exactly one rule of the form

$$i(zj) \to \rho$$

with $\rho \in RHS(A_s, A_i, \Delta, \sigma)$.

For every $G_s \subseteq A_s$, $G_i \subseteq A_i$, and $\sigma \in \Sigma \cup \{root\}$ with $rank(\sigma) = k \geq 0$, the set of *σ-right-hand sides over G_s, G_i and Δ*, denoted by $RHS(G_s, G_i, \Delta, \sigma)$, is the smallest subset RHS of $(G_s \cup G_i \cup \Delta \cup [k] \cup \{z, (,), ,\})^*$ such that the following three conditions hold:

(i) For every $\delta \in \Delta^{(r)}$ with $r \geq 0$, and $\rho_1, \ldots, \rho_r \in RHS$, the tree $\delta(\rho_1, \ldots, \rho_r) \in RHS$.

(ii) For every $s \in G_s$, $j \in [k]$, the tree $s(zj) \in RHS$.

(iii) For every $i \in G_i$, the tree $i(z) \in RHS$. □

For an *si*-tree transducer $M = (A, \Delta, \Sigma, s_{in}, root, R)$, we fix the following notions and notations.

- The set $\Sigma \cup \{root\}$ is denoted by Σ_+.
- In the rules of R, the symbol z is called *path variable*.
- For every $\sigma \in \Sigma^{(k)}$, the set of *inside attribute occurrences* of σ, denoted by $in(\sigma)$, is the set $\{s(z) \mid s \in A_s\} \cup \{i(zj) \mid i \in A_i, j \in [k]\}$. The set of *inside attribute occurrences* of *root*, denoted by $in(root)$, is the set $\{s_{in}(z)\} \cup \{i(z1) \mid i \in A_i\}$. The set of *outside attribute occurrences* of σ, denoted by $out(\sigma)$, is the set $\{i(z) \mid i \in A_i\} \cup \{s(zj) \mid s \in A_s, j \in [k]\}$. The set of *outside attribute occurrences* of *root*, denoted by $out(root)$, is the set $\{s(z1) \mid s \in A_s\}$. The set of *attribute occurrences* of $\sigma \in \Sigma_+$, denoted by $att(\sigma)$, is the set $in(\sigma) \cup out(\sigma)$.

- For $a \in A$, $\sigma \in \Sigma_+^{(k)}$ and $\eta \in \{zj \mid j \in [k] \cup \{\varepsilon\}\}$, we call a rule of R_σ with the left-hand side $a(\eta)$ an (a, η, σ)-rule. The right-hand side of this rule is denoted by $rhs(a, \eta, \sigma)$. We note that only outside attribute occurrences of σ appear in $rhs(a, \eta, \sigma)$ and that for every $a(\eta) \in in(\sigma)$, there is exactly one (a, η, σ)-rule in R.

Example 2.2 We define the *si*-tree transducer $M_1 = (\mathcal{A}, \Delta, \Sigma, s, root, R)$ with:
$\Delta = \{B^{(1)}, T^{(2)}, L^{(1)}, R^{(1)}, E^{(0)}\}$,
$\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$,
$\mathcal{A} = (A, A_s, A_i)$ with $A = \{s, i\}$, $A_s = \{s\}$, and $A_i = \{i\}$, and
$R = R_{root} \cup R_\sigma \cup R_\alpha$ is the following set of rules:

$$
\begin{array}{lcll}
R_{root} = & \{s(z) & \to B(s(z1)), & (1) \\
& i(z1) & \to E \quad \} & (2) \\
R_\sigma = & \{s(z) & \to T(s(z1), s(z2)), & (3) \\
& i(z1) & \to L(i(z)), & (4) \\
& i(z2) & \to R(i(z)) \quad \} & (5) \\
R_\alpha = & \{s(z) & \to B(i(z)) \quad \} & (6)
\end{array}
$$

The *si*-tree transducer M_1 takes a binary tree e over the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$ as argument and it delivers a tree t which has the same structure as e, but in which every leaf node n is substituted by an encoding of the reverse path from the root of e to n. The encoding of a reverse path is a monadic tree over the ranked alphabet $\{B^{(1)}, L^{(1)}, R^{(1)}, E^{(0)}\}$, where the symbol L (and R) represent the left son (and the right son, respectively) of a node and the symbol B (and E) is the first symbol (and the last symbol, respectively) of each path encoding (cf. Figure 2). □

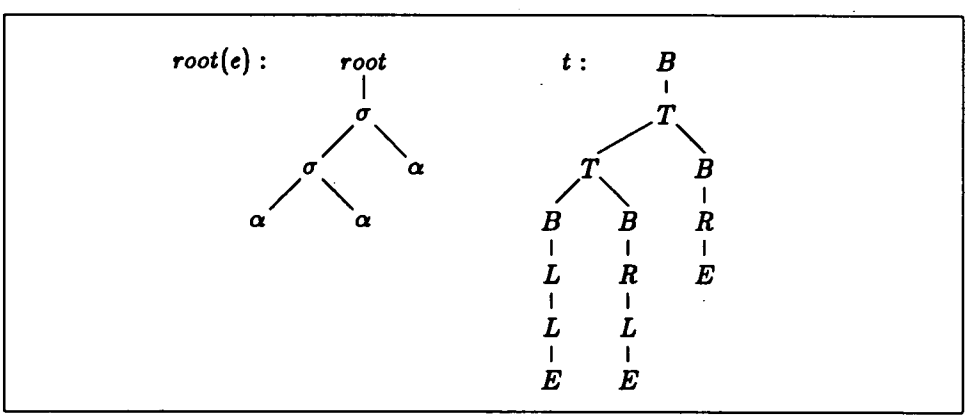


Figure 2: The control tree $\tilde{e}$ and the calculated output tree t.

Observation 2.3

1. Top–down tree transducers [Rou70,Tha70,Eng75] are *si*–tree transducers without inherited attributes.

2. Attributed tree transducers [Fül81] are *si*–tree transducers in which, for every inherited attribute i, the right–hand side of the $(i, z1, root)$–rule is a tree over Δ. In accordance to [Gie88] *si*–tree transducers are *full attributed tree transducers*. But in the sequel we also use simply the notion attributed tree transducer. □

Before working out the definition of the derivation relation, we first introduce a uniform classification scheme for subclasses of *si*–tree transducers which are induced by the number of attributes.

Definition 2.4

- Let $k_s \in \mathbb{N} - \{0\}$ and $k_i \in \mathbb{N}$. An $s_{(k_s)}i_{(k_i)}$*–tree transducer* M is an *si*–tree transducer with at most k_s synthesized attributes and with at most k_i inherited attributes.

- An *s–tree transducer* is an $s_{(k_s)}i_{(0)}$–tree transducer for some $k_s \in \mathbb{N} - \{0\}$, i.e., an *si*–tree transducer without inherited attributes. □

In the next definition we inductively describe the set of all sentential forms of attributed tree transducers. For a given control tree $\tilde{e} = root(e)$ with $e \in T\langle\Sigma\rangle$, a sentential form is a tree over attributes, working symbols, and paths through $\tilde{e}$. Moreover, the argument of an attribute is always a path through $\tilde{e}$ and vice versa a path may only occur in the argument of an attribute.

Definition 2.5 Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes. Moreover, let $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$ and let Δ' be a ranked alphabet with $\Delta \subseteq \Delta'$. The set of $(A, s_{in}, paths(\tilde{e}), \Delta')$*–sentential forms*, denoted by $SF(A, s_{in}, paths(\tilde{e}), \Delta')$, is defined inductively as follows where we abbreviate $SF(A, s_{in}, paths(\tilde{e}), \Delta')$ by SF.

(i) For every $\delta \in \Delta'^{(r)}$ with $r \geq 0$ and $t_1, \ldots, t_r \in SF$, the tree $\delta(t_1, \ldots, t_r) \in SF$.

(ii) For every $a \in A$ and $p \in paths(\tilde{e})$ with $p \neq \varepsilon$, the tree $a(p) \in SF$.

(iii) The tree $s_{in}(\varepsilon) \in SF$. □

Notice that the tree $\tilde{e}$ does not occur in sentential forms. It is only needed to define the set of paths of $\tilde{e}$.

For an attributed tree transducer $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ with system $\mathcal{A} = (A, A_s, A_i)$ of attributes and for a tree $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$, the set of *attribute occurrences* of $\tilde{e}$, denoted by $att(\tilde{e})$, is the set $\{s_{in}(\varepsilon)\} \cup \{a(p) \mid a \in A,$

$p \in paths(\tilde{e}), p \neq \epsilon\}$. If $\tilde{e} = root(e)$ for a particular tree $e \in T\langle\Sigma\rangle$, then we define $att(e) = att(\tilde{e}) - \{s_{in}(e)\}$.

Let $e' \in T\langle\Sigma_+ \cup \{w\}\rangle$ with exactly one occurrence of a symbol $w \notin \Sigma_+$ be a pattern in a control tree $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$, such that $e' = subtree(\tilde{e}[p' \leftarrow w], p)$ holds for some paths $p, p' \in paths(\tilde{e})$. The set of *inside attribute occurrences of e' with respect to $\tilde{e}$* is the set $(\{s(p) \mid s \in A_s\} \cup \{i(p') \mid i \in A_i\}) \cap att(\tilde{e})$. The set of *outside attribute occurrences of e' with respect to $\tilde{e}$* is the set $(\{i(p) \mid i \in A_i\} \cup \{s(p') \mid s \in A_s\}) \cap att(\tilde{e})$. (The intersection with $att(\tilde{e})$ is necessary to handle the case $p = \epsilon$.) If the underlying control tree $\tilde{e}$ is clear from the context, then we simply use the notions inside and outside attribute occurrences of e'.

Now we describe the derivation relation of an attributed tree transducer M with respect to a control tree $\tilde{e}$. For later purposes, we restrict the derivation relation to work only on particular parts of $\tilde{e}$ parameterizing the derivation relation with a subset $P \subseteq paths(\tilde{e})$.

Definition 2.6 Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes. Let $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$ and $P \subseteq paths(\tilde{e})$. The *derivation relation of M with respect to $\tilde{e}$ and P*, denoted by $\Rightarrow_{M,\tilde{e},P}$, is a binary relation on $SF(A, s_{in}, paths(\tilde{e}), \Delta)$ defined as follows:
For every $t_1, t_2 \in SF(A, s_{in}, paths(\tilde{e}), \Delta)$, $t_1 \Rightarrow_{M,\tilde{e},P} t_2$, iff

- there is a $t' \in SF(A, s_{in}, paths(\tilde{e}), \Delta \cup \{u\})$ in which the 0–ary symbol $u \notin A \cup \Delta$ occurs exactly once,
- there is an attribute $a \in A$,
- there is a path $p \in paths(\tilde{e})$,

such that $t_1 = t'[u/a(p)]$ and if one of the following two conditions holds:

1. - a is a synthesized attribute,
   - $p \in P$ and $label(\tilde{e}, p) = \sigma$ for some $\sigma \in \Sigma_+^{(k)}$ with $k \geq 0$,
   - there is a rule $a(z) \rightarrow \rho$ in R_σ, and
   - $t_2 = t'[u/\rho[z/p]]$.
2. - a is an inherited attribute,
   - $p = p'j$ for some $p' \in P$, $label(\tilde{e}, p') = \sigma$ for some $\sigma \in \Sigma_+^{(k)}$ with $k \geq 1$, and $j \in [k]$,
   - there is a rule $a(zj) \rightarrow \rho$ in R_σ, and
   - $t_2 = t'[u/\rho[z/p']]$. □

Note that in case 2. the path p itself needs not to be in P. This is important for the later construction in the pumping lemma. If M or $\tilde{e}$ are known from the context, we drop the corresponding indices from $\Rightarrow$. If $P = paths(\tilde{e})$, then we drop P.

Before presenting an example derivation we have to explain the special role of the marker *root*. It allows us to handle the calculation of the values of inherited attribute occurrences at the root of an input tree e like all the other attribute occurrences of e. Taking the control tree $root(e)$, we can specify the value of an inherited attribute occurrence at the root of e by a rule in R_{root}. In particular, the inherited attribute occurrences at the root of e may depend on the synthesized attribute occurrences at the root of e. This mechanism has also been used in [KV94]. It is more general than the solution presented in [Fül81], where special trees in $T\langle\Delta\rangle$ are used to specify the values of the inherited attribute occurrences at the root of e.

Example 2.7 Let M_1 be the attributed tree transducer defined in Example 2.2 and let $\tilde{e} = root(\sigma(\sigma(\alpha,\alpha),\alpha))$ be the control tree. We abbreviate $\Rightarrow_{M_1,\tilde{e},paths(\tilde{e})}$ by $\Rightarrow$. The number of the applied rule is indicated as a subscript. The control tree and the calculated output tree are also shown in Figure 2.

$$
\begin{aligned}
& s(e)\\
\Rightarrow_{(1)}\ & B(s(1))\\
\Rightarrow_{(3)}\ & B(T(s(11),s(12)))\\
\Rightarrow_{(3)}\ & B(T(T(s(111),s(112)),s(12)))\\
\Rightarrow_{(6)}\ & B(T(T(B(i(111)),s(112)),s(12)))\\
\Rightarrow_{(4)}\ & B(T(T(B(L(i(11))),s(112)),s(12)))\\
\Rightarrow_{(4)}\ & B(T(T(B(L(L(i(1)))),s(112)),s(12)))\\
\Rightarrow_{(2)}\ & B(T(T(B(L(L(E))),s(112)),s(12)))\\
\Rightarrow^{+}\ & B(T(T(B(L(L(E))),B(R(L(E)))),B(R(E))))
\end{aligned}
$$

□

2.4 Noncircular attributed tree transducers

Since an attributed tree transducer can be circular (in the same sense as an attribute grammar), we can conclude that, in general, the derivation relations of attributed tree transducers are not noetherian (cf., e.g., [Ems91] for an example of a circular attributed tree transducer.) However, noncircular attributed tree transducers induce noetherian derivation relations. The notion of circularity is taken from [Fül81]:

Definition 2.8 Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an si–tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes.

1. M is *circular* if
   - there is an $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$
   - there is an $a(p) \in SF(A, s_{in}, paths(\tilde{e}), \Delta)$ with $a \in A$ and $p \in paths(\tilde{e})$,
   - there is a $t \in SF(A, s_{in}, paths(\tilde{e}), \Delta \cup \{u\})$ in which the 0–ary symbol $u \notin A \cup \Delta$ occurs exactly once,

such that $a(p) \Rightarrow^{+}_{M,\tilde{e}} t[u/a(p)]$.

2. M is *noncircular* if it is not circular. □

For the definition of the tree transformation computed by an attributed tree transducer we use the following result (cf. Theorem 3.17 of [KV94]).

Lemma 2.9 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*-tree transducer. If M is noncircular, then for every $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$, the relation $\Rightarrow_{M,\tilde{e}}$ is confluent and noetherian. □

Since the derivation relations of noncircular attributed tree transducers are confluent and noetherian, every sentential form has a unique normal form. This is the basis for the definition of the tree transformation which is computed by an attributed tree transducer.

Definition 2.10 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be a noncircular *si*- tree transducer. The *tree transformation computed by* M, denoted by $\tau(M)$, is the total function of type $T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ defined as follows. For every $e \in T\langle\Sigma\rangle$,

$$\tau(M)(e) = nf(\Rightarrow_{M,root(e)}, s_{in}(e)).$$ □

In the rest of this paper, we always mean noncircular attributed tree transducers when we talk about attributed tree transducers.

For a given control tree $\tilde{e}$, for a given derivation $s_{in}(e) \Rightarrow^{+}_{\tilde{e}} t$ (abbreviated by d), where $t = nf(\Rightarrow_{\tilde{e}}, s_{in}(e))$, and for a given path p in $\tilde{e}$ we define the set $attset(d,p)$ of those attributes a, for which there are attribute occurrences $a(p)$ in a sentential form during the derivation d. This concept is the same as the concept of *state-set* described in [ERS80], however, we use another way of definition.

Definition 2.11 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*-tree transducer with system $A = (A, A_s, A_i)$ of attributes. Let $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$. Let d be the derivation $s_{in}(e) = t_0 \Rightarrow_{\tilde{e}} t_1 \Rightarrow_{\tilde{e}} \ldots \Rightarrow_{\tilde{e}} t_n = nf(\Rightarrow_{\tilde{e}}, s_{in}(e))$ with $n \geq 1$ derivation steps, and let $p \in paths(\tilde{e})$. Then we define the *attribute-set of* d *and* p, denoted by $attset(d,p)$, by

$$\bigcup_{j=0}^{n} attset'(t_j, p) \qquad \text{where}$$

$attset' : SF(A, s_{in}, paths(\tilde{e}), \Delta) \times paths(\tilde{e}) \longrightarrow \mathcal{P}(A)$ is defined as follows:
For every $\delta \in \Delta^{(r)}$, $r \geq 0$, $t_1, \ldots, t_r \in SF(A, s_{in}, paths(\tilde{e}), \Delta)$, $p \in paths(\tilde{e})$,
$attset'(\delta(t_1, \ldots, t_r), p) = \bigcup_{j=1}^{r} attset'(t_j, p)$.
For every $a(p') \in att(\tilde{e})$, $p \in paths(\tilde{e})$, if $p = p'$, then
$attset'(a(p'), p) = \{a\}$.
For every $a(p') \in att(\tilde{e})$, $p \in paths(\tilde{e})$, if $p \neq p'$, then
$attset'(a(p'), p) = \emptyset$. □

Example 2.12 Let M_1 be the attributed tree transducer defined in Example 2.2 and let $\tilde{e} = root(\alpha)$ be the control tree.
Let $d = (s(e) \Rightarrow_{\tilde{e}} B(s(1)) \Rightarrow_{\tilde{e}} B(B(i(1))) \Rightarrow_{\tilde{e}} B(B(E)))$ be a derivation.
Then $attset(d, e) = attset'(s(e), e) = \{s\}$
and $attset(d, 1) = attset'(B(s(1)), 1) \cup attset'(B(B(i(1))), 1) = \{s, i\}$ hold. □

In fact, the attribute–set of a path does not depend on the chosen derivation.

Lemma 2.13 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer. Let d_1 and d_2 be two derivations $s_{in}(e) \Rightarrow^+_{\tilde{e}} nf(\Rightarrow_{\tilde{e}}, s_{in}(e))$ for some $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$. Then, for every path $p \in paths(\tilde{e})$, the sets $attset(d_1, p)$ and $attset(d_2, p)$ are equal. □

Definition 2.14 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer. Let $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$ and let $p \in paths(\tilde{e})$. The *attribute–set of* $\tilde{e}$ *and* p, denoted by $attset(\tilde{e}, p)$, is the set $attset(d, p)$ for some derivation $d = (s_{in}(e) \Rightarrow^+_{\tilde{e}} nf(\Rightarrow_{\tilde{e}}, s_{in}(e)))$. □

2.5 Producing and visiting attributed tree transducers

The pumping lemma in the next section is only valid for special kinds of attributed tree transducers. In the following definition we introduce the concepts of producing (every rule application produces at least one new output symbol), and visiting (every node of a control tree is visited by at least one attribute) tree transducers.

Definition 2.15 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer. M is

- *producing*, if, for every rule $\lambda \to \rho$ in R, the size of ρ with respect to Δ is at least 1, i.e., $size_\Delta(\rho) \geq 1$,
- *visiting*, if, for every control tree $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$ and for every $p \in paths(\tilde{e})$, the attribute–set of $\tilde{e}$ and p is not empty, i.e., $attset(\tilde{e}, p) \neq \emptyset$.

□

In the rest of this paper we always mean producing and visiting (and noncircular) attributed tree transducers, when we talk about attributed tree transducers. We denote the *classes of tree transformations computed* by (noncircular, producing, and visiting) *si*–tree transducers, $s_{(k_s)}i_{(k_i)}$– tree transducers, and *s*–tree transducers by SIT, $S_{(k_s)}I_{(k_i)}T$, and ST, respectively.

2.6 Output languages of attributed tree transducers

The pumping lemma which we introduce in the next section, deals with output languages of tree transformations of attributed tree transducers. The output language of a tree transformation τ is defined as the range of τ.

Definition 2.16 Let $\tau : T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ be a tree transformation. The *output language of* τ, denoted by $L_{out}(\tau)$ is defined as follows:

$$L_{out}(\tau) = \{t \in T\langle\Delta\rangle \mid \text{there is an } e \in T\langle\Sigma\rangle \text{ such that } \tau(e) = t\}. \qquad \square$$

If $\tau(M)$ is a tree transformation computed by an attributed tree transducer M, we simply write $L_{out}(M)$ instead of $L_{out}(\tau(M))$ and we simply call $L_{out}(M)$ the *output language of* M instead of the output language of the tree transformation computed by M.

We denote the *classes of output languages* of (noncircular, producing, and visiting) si–tree transducers, $s_{(k_s)}i_{(k_i)}$–tree transducers, and s–tree transducers by SIT_{out}, $S_{(k_s)}I_{(k_i)}T_{out}$, and ST_{out}, respectively.

If we want to prove that a certain tree transformation τ is not an element of the class SIT, then the output language $L_{out}(\tau)$ can be very useful. It would suffice to show with the help of the pumping lemma presented in the next section that $L_{out}(\tau) \notin SIT_{out}$. Thus, since $L_{out}(\tau)$ is not the range of an si–tree transducer, τ cannot be the tree transformation computed by an si–tree transducer.

For the sake of convenience, we now omit the parantheses for arguments of monadic output symbols in the rest of the paper; the parantheses for arguments of attributes remain.

Example 2.17 Let M_1 be the attributed tree transducer defined in Example 2.2 and let d be the derivation of Example 2.7.

Thus, in the following we write rule (1) of M_1 in the form $s(z) \rightarrow B\,s(z1)$. Note that there are still parantheses in the attribute occurrence $s(z1)$. The notation $s(z) \rightarrow T(s(z1), s(z2))$ of rule (3) is left unchanged, because T is a binary output symbol.

In anology we write the last but one sentential form of d that was shown in Example 2.7 as $B\,T(T(B\,L\,L\,E, s(112)), s(12))$. $\square$

3 Pumping lemma for attributed tree transducers

Before presenting the pumping lemma for si–tree transducers and working out the proof formally, we want to illustrate the central idea and show an example. Although the pumping lemma only deals with output trees and not with the control trees corresponding to them via a tree transformation, the control trees play an important part.

Let M be an attributed tree transducer. If we choose a sufficiently large output tree t, then every control tree $\tilde{e} = root(e)$ with $\tau(M)(e) = t$ is high enough, such that it has a path p, on which two different nodes x_1 and x_2 can be found such that (cf. Figure 3)

- there exist strings p_1, p_2, and p_3 such that $|p_2| > 0$ and $p = p_1p_2p_3$,

- x_1 and x_2 can be reached from the root by p_1 and p_1p_2, respectively, i.e., $x_1 = node(\tilde{e}, p_1)$ and $x_2 = node(\tilde{e}, p_1p_2)$, and
- the attribute–sets $attset(\tilde{e}, p_1)$ and $attset(\tilde{e}, p_1p_2)$ are equal.

These two nodes define a decomposition of $\tilde{e}$ into three input patterns e', e'', and e'''. Intuitively,

- e' is the tree $\tilde{e}$ without the subtree which has x_1 as root.
- e'' is the tree which has x_1 as root without the subtree which has x_2 as root.
- e''' is the tree which has x_2 as root.

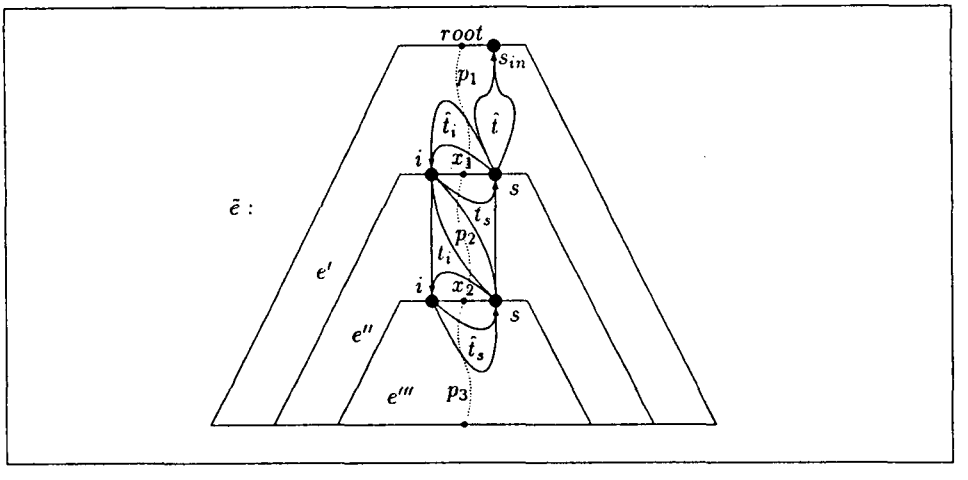


Figure 3: Control tree $\tilde{e}$ with input patterns and induced output patterns.

This decomposition of the control tree $\tilde{e}$ induces a decomposition of the output tree t into a certain output pattern $\hat{t}$, certain output patterns t_s and $\hat{t}_s$ for every synthesized attribute s, and certain output patterns t_i and $\hat{t}_i$ for every inherited attribute i. Roughly speaking, these patterns correspond to normal forms of certain attribute occurrences of the patterns e', e'', and e'''. More precisely,

- The tree $\hat{t}$ corresponds to the normal form of $s_{in}(\varepsilon)$ that is calculated only on the nodes of e'.
- For every synthesized attribute s in the attribute–set of the two relevant nodes x_1 and x_2, the tree t_s (and $\hat{t}_s$) corresponds to the normal form of $s(p_1)$ (and $s(p_1p_2)$, respectively) that is calculated only on the nodes of e'' (and e''', respectively).

- For every inherited attribute i in the attribute- -set of the two relevant nodes x_2 and x_1, the tree t_i (and $\hat{t}_i$) corresponds to the normal form of $i(p_1p_2)$ (and $i(p_1)$, respectively) that is calculated only on the nodes of e'' (and e', respectively).

In Figure 3 these output patterns are indicated; the root of every output pattern is represented by an arrow. The reader should not be misleaded by the cycles among the pieces of the final output tree: we consider noncircular attributed tree transducers and, only for the sake of simplicity of the figure, we show only one inherited attribute and one synthesized attribute; thus, dependencies are folded and suggest cycles which are not there.

If we construct new control trees by repeating the pattern e'' arbitrarily often, then we can get new output trees by translating the new control trees. All of them are by definition elements of $L_{out}(M)$. The output patterns t_s and t_i must be used for every repetition of e'' to obtain the new output tree. Figure 4 shows the situation in which e'' is repeated twice.

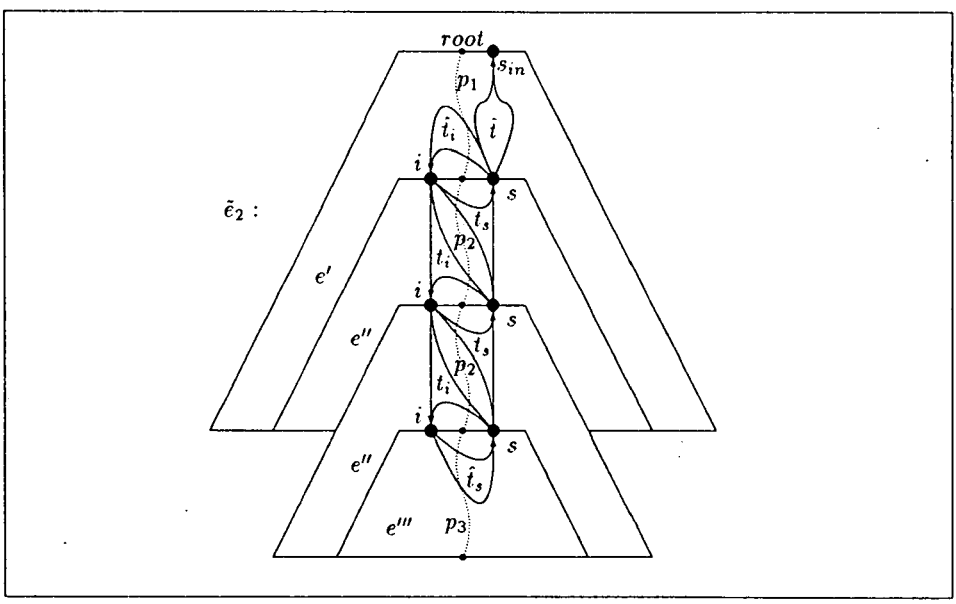


Figure 4: Control tree with two repetitions of e'' and output patterns.

In the pumping lemma we use a recursive function $\underline{tree'}$ which walks through the patterns of the control tree and builds up the output using the output patterns

defined above.

Note that for the pumping process it is not necessary that the nodes x_1 and x_2 are labeled by the same symbol, in contrast to the pumping lemma for context–free languages (cf. for example [BPS61]). This is due to the fact that we only deal with ranked alphabets rather than heterogeneous signatures; thus only the rank of the symbols is important when building up trees.

We show the input patterns, the output patterns and the pumping process in the following example.

Example 3.1 Let M_1 be the *si*–tree transducer defined in Example 2.2. For simplicity we repeat the rules of M_1, omitting superfluous paranthesis:

$$
\begin{array}{lcll}
R_{root} & = & \{s(z) & \rightarrow B\, s(z1), \\
 & & i(z1) & \rightarrow E \qquad\qquad\quad \} \\
R_\sigma & = & \{s(z) & \rightarrow T(s(z1), s(z2)), \\
 & & i(z1) & \rightarrow L\, i(z), \\
 & & i(z2) & \rightarrow R\, i(z) \qquad\quad \} \\
R_\alpha & = & \{s(z) & \rightarrow B\, i(z) \qquad\quad \}
\end{array}
$$

Although the pumping lemma only guarantees to work with an output tree t with $size(t) \geq n_{M_1}$ for a certain natural number n_{M_1} (which is called the pumping index of M_1), it often also works for smaller output trees. Nevertheless, the pumping index is needed in the proof of the pumping lemma. In this example we have $n_{M_1} = 2^{15}$. The reader can check this after having read Definition 3.2.

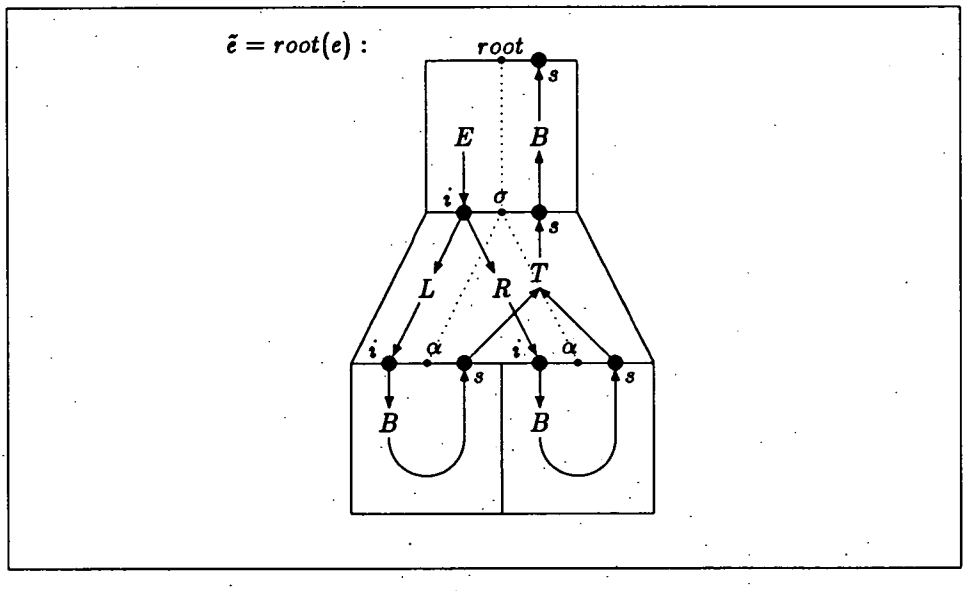


Figure 5: Control tree $\tilde{e}$ with right–hand sides of rules.

Here we take the smaller tree $t = nf(\Rightarrow_{\tilde{e}}, s_{in}(e))$, where $\tilde{e} = root(\sigma(\alpha, \alpha))$ is the control tree. In Figure 5 the control tree $\tilde{e}$ is shown by dotted lines, where additionally the right-hand sides of those rules are incorporated which are necessary to compute the values of the attribute occurrences of $\tilde{e}$.

Now we consider the two nodes $node(\tilde{e}, 1)$ and $node(\tilde{e}, 11)$ of the control tree $\tilde{e}$ which can be reached from the root of $\tilde{e}$ by paths 1 and 11. Note that $\sigma = label(\tilde{e}, 1)$, $\alpha = label(\tilde{e}, 11)$, and $attset(\tilde{e}, 1) = attset(\tilde{e}, 11) = \{s, i\}$. In this case we have chosen the path $p = 11$ with its subpaths $p_1 = 1$, $p_2 = 1$, and $p_3 = \varepsilon$. In Figure 6 we show three patterns in $\tilde{e}$ with the nodes reached by the paths ε, 1, and 11, respectively, as roots. Again the right-hand sides of rules are incorporated into the figure.

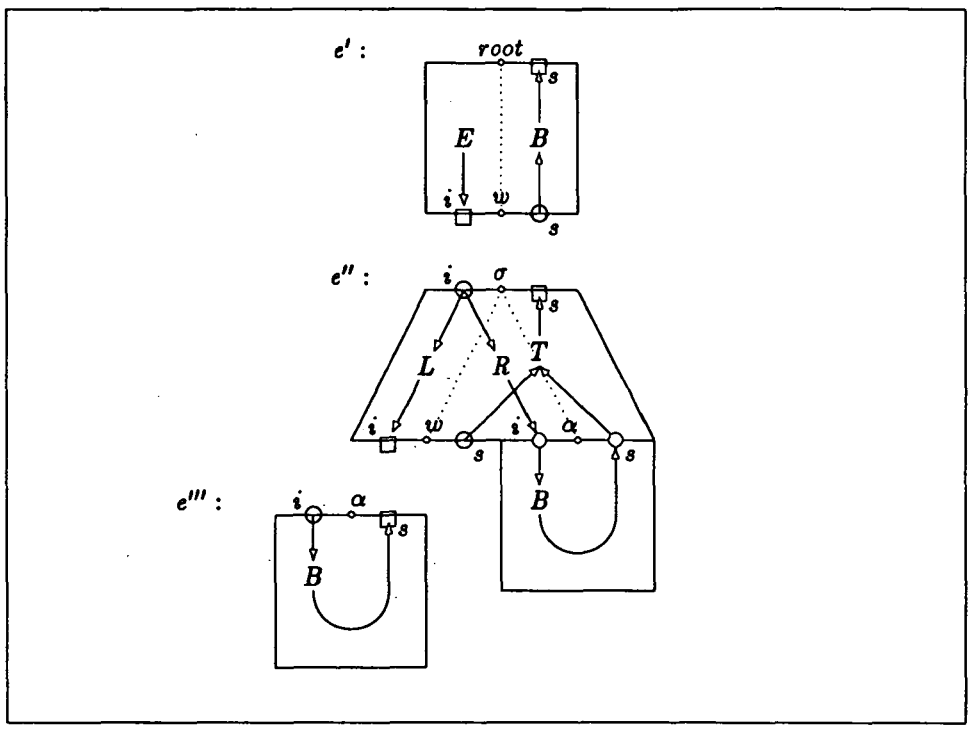


Figure 6: Input patterns e', e'' and e''' with right-hand sides of rules.

For later purposes, in Figure 7 we also show the control tree $\tilde{e}$ and the patterns e', e'', and e''' framing those parts of the patterns which only consist of input symbols. In fact, we have $\tilde{e} = e'[w/e''[w/e''']]$.

With these preparations we can obtain the patterns in the output tree t as follows: Roughly speaking, for each of the patterns e', e'', and e''', we calculate the values of the inside attribute occurrences as function in the values of the outside attribute occurrences. Therefore we can use the dependencies among the attribute occurrences presented in Figure 6, where the outside attribute occurrences and the inside attribute occurrences are depicted as non-filled cycles and non- filled boxes,

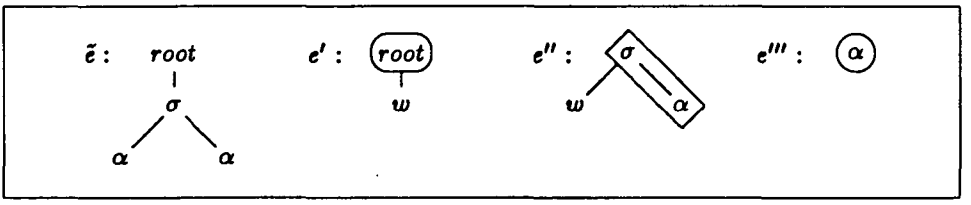


Figure 7: Control tree $\tilde{e}$ and its decomposition.

respectively, whereas the other attribute occurrences are depicted as filled cycles. More precisely, we calculate

- the values $\hat{t}$ and $\hat{t}_i$ of the inside attribute occurrences $s(\varepsilon)$ and $i(1)$ of e', respectively, as function in the value of the outside attribute occurrence $s(1)$ of e',
- the values t_s and t_i of the inside attribute occurrences $s(1)$ and $i(11)$ of e'', respectively, as function in the values of the outside attribute occurrences $s(11)$ and $i(1)$ of e'',
- and the value $\hat{t}_s$ of the inside attribute occurrence $s(11)$ of e''' as function in the value of the outside attribute occurrence $i(11)$ of e''',

and replace the synthesized attribute occurrences $s(1)$ and $s(11)$ by the symbol s with rank 0 and the inherited attribute occurrences $i(1)$ and $i(11)$ by the symbol i with rank 0. For the sake of understanding we choose exactly the attributes as names for the new symbols. Based on the rank, the reader can retrieve whether symbols or attributes are concerned at a time. The values of the output patterns are as follows:

$$
\begin{array}{lcrcl}
\hat{t} & = & nf(\Rightarrow_{\tilde{e},\{\varepsilon\}}, s(\varepsilon))[s(1)/s] & = & B\ s, \\
\hat{t}_i & = & nf(\Rightarrow_{\tilde{e},\{\varepsilon\}}, i(1))[s(1)/s] & = & E, \\
t_s & = & nf(\Rightarrow_{\tilde{e},\{1,12\}}, s(1))[s(11)/s,\ i(1)/i] & = & T(s, B\ R\ i), \\
t_i & = & nf(\Rightarrow_{\tilde{e},\{1,12\}}, i(11))[s(11)/s,\ i(1)/i] & = & L\ i, \\
\hat{t}_s & = & nf(\Rightarrow_{\tilde{e},\{11\}}, s(11))[i(11)/i] & = & B\ i.
\end{array}
$$

In Figure 8 we show the output tree t and the output patterns defined above. For later purposes we also frame the parts of the patterns which only consist of output symbols.

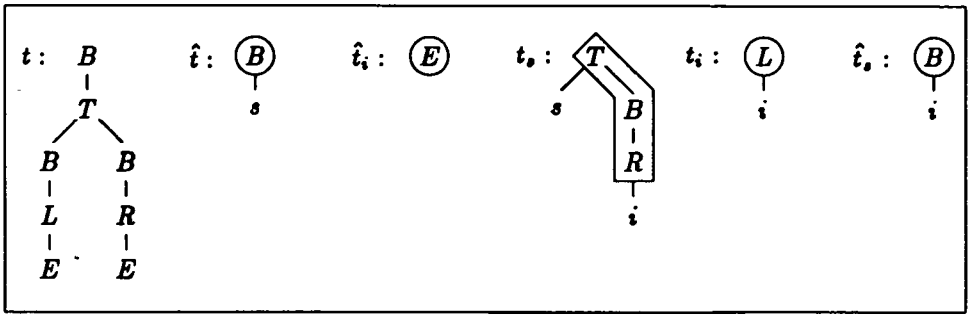


Figure 8: Output tree t and output patterns.

Now we show the pumping process in the cases in which

(i) the pattern e'' is dropped ($r = 0$),

(ii) the pattern e'' occurs once ($r = 1$), and

(iii) the pattern e'' occurs twice ($r = 2$).

Thus we have the control tree

(i) $\tilde{e}_0 = e'[w/e''']$, if $r = 0$,

(ii) $\tilde{e} = \tilde{e}_1 = e'[w/e''[w/e''']]$, if $r = 1$, and

(iii) $\tilde{e}_2 = e'[w/e''[w/e''[w/e''']]]$, if $r = 2$.

For every $0 \leq r \leq 2$, the normal form $nf(\Rightarrow_{\tilde{e}_r}, s_{in}(e))$ is denoted by $\underline{tree}(r)$. It can also be calculated using the above defined patterns of t as follows:

We start with the pattern $\hat{t} = Bs$ that corresponds to the attribute occurrence $s(\varepsilon)$, and replace the symbol s by the function call $\underline{tree'}(s, r, 1)$. Roughly speaking, the recursive function $\underline{tree'}$ moves through the different patterns of $\tilde{e}_r$ and it constructs the output using the output patterns. Every function call of $\underline{tree'}$ delivers one output pattern, in which the symbols s and i are replaced by new function calls of $\underline{tree'}$.

The function $\underline{tree'}$ has three parameters. The first parameter is one of the symbols s or i. It indicates, whether we have to use one of the patterns t_s or $\hat{t}_s$ (in case of the symbol s), or one of the patterns t_i or $\hat{t}_i$ (in case of the symbol i). The other two parameters are natural numbers. The second parameter r indicates the number of repetitions of e'' in the control tree $\tilde{e}_r$. It is constant during the calculation of a certain output tree. The third parameter l indicates the level of the input pattern, where $\underline{tree'}$ currently works. $l = 0$ means the pattern e', $1 \leq l \leq r$ means the l–th repetition of the pattern e'', and $l = r + 1$ means the pattern e'''.

If $1 \leq l \leq r$, then $\underline{tree'}$ uses the pattern $t_s = T(s, BRi)$ (or $t_i = Li$); this pattern corresponds to the normal form which is calculated only on the nodes of the pattern e'' starting with the attribute occurrence $s(1)$ (or $i(11)$, respectively).

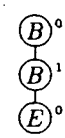
If $l = r+1$, then $\underline{tree'}$ uses the pattern $\hat{t}_s = Bi$; this pattern corresponds to the normal form which is calculated only on the nodes of the pattern e''' starting with the attribute occurrence $s(11)$.

If $l = 0$, then $\underline{tree'}$ uses the pattern $\hat{t}_i = E$; this pattern corresponds to the normal form which is calculated only on the nodes of the pattern e' starting with the attribute occurrence $i(1)$.

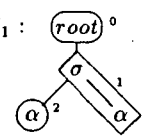
If l is the current level of the function $\underline{tree'}$, then every occurrence of the symbol s (or i) in the produced output pattern is replaced by a function call $\underline{tree'}(s, r, l+1)$ (or $\underline{tree'}(i, r, l-1)$, respectively), because $\underline{tree'}$ has to move one level down (or up, respectively) in $\tilde{e}_r$.

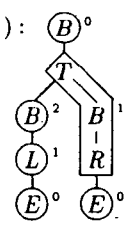
$\underline{tree}(0)$
$= B\ \underline{tree'}(s,0,1)$
$= B\ B\ \underline{tree'}(i,0,0)$
$= B\ B\ E$

$\tilde{e}_0$: 

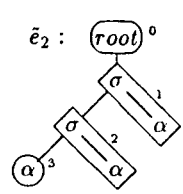
$\underline{tree}(0)$: 

$\underline{tree}(1)$
$= B\ \underline{tree'}(s,1,1)$
$= B\ T(\underline{tree'}(s,1,2), B\ R\ \underline{tree'}(i,1,0))$
$= B\ T(B\ \underline{tree'}(i,1,1), B\ R\ E)$
$= B\ T(B\ L\ \underline{tree'}(i,1,0), B\ R\ E)$
$= B\ T(B\ L\ E, B\ R\ E)$

$\tilde{e} = \tilde{e}_1$: 

$t = \underline{tree}(1)$: 

$\underline{tree}(2)$
$= B\ \underline{tree'}(s,2,1)$
$= B\ T(\underline{tree'}(s,2,2), B\ R\ \underline{tree'}(i,2,0))$
$= B\ T(T(\underline{tree'}(s,2,3),$
$\quad B\ R\ \underline{tree'}(i,2,1)), B\ R\ E)$
$= B\ T(T(B\ \underline{tree'}(i,2,2),$
$\quad B\ R\ L\ \underline{tree'}(i,2,0)), B\ R\ E)$
$= B\ T(T(B\ L\ \underline{tree'}(i,2,1),$
$\quad B\ R\ L\ E), B\ R\ E)$
$= B\ T(T(B\ L\ L\ \underline{tree'}(i,2,0),$
$\quad B\ R\ L\ E), B\ R\ E)$
$= B\ T(T(B\ L\ L\ E, B\ R\ L\ E), B\ R\ E)$

$\tilde{e}_2$: 

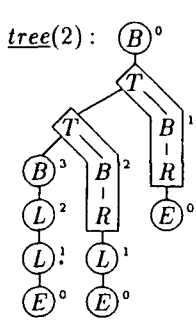
$\underline{tree}(2)$: 

Figure 9: Calculations of $\underline{tree}(r)$ for $0 \leq r \leq 2$ and decompositions of $\tilde{e}_r$ and $\underline{tree}(r)$.

In Figure 9 we show besides the calculations of the output trees $\underline{tree}(0)$, $\underline{tree}(1)$, and $\underline{tree}(2)$, their decompositions into the output patterns. Every output pattern is labeled with the level $0 \leq l \leq r+1$ of the input pattern which causes it. We also show the control trees, corresponding to the output trees, and their decompositions into input patterns which are labeled with their level. □

As stated in the last example, the pumping process only guarantees to work for output trees which are large enough. This requirement is satisfied, if the size of the output tree is at least the pumping index of the given attributed tree transducer. Recall that we only consider noncircular, producing, and visiting attributed tree transducers.

Definition 3.2 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with k_s synthesized and k_i inherited attributes. We define

$$\begin{aligned} c_M &= max\{size_A(\rho) \mid (\lambda \to \rho) \in R\} \\ &\quad \text{(maximum number of attribute occurrences in right-hand sides)}, \\ l_M &= max\{size_\Delta(\rho) \mid (\lambda \to \rho) \in R\} \\ &\quad \text{(maximum number of output symbols in right-hand sides)}, \\ m_M &= max\{rank(\sigma) \mid \sigma \in \Sigma\} \\ &\quad \text{(maximum rank of input symbols)}, \end{aligned}$$

and the *pumping index* n_M *of* M as:

$$n_M = 1 + l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j \quad \text{where} \quad n'_M = \sum_{j=0}^{2^{k_i}\cdot(2^{k_s}-1)} (m_M)^j.$$

□

In the proof of the pumping lemma we need the fact that the subtree e of a control tree $root(e)$ has at least some particular height; the desired height is $2^{k_i} \cdot (2^{k_s}-1)+2$ (cf. the proof of Theorem 3.4 for an argumentation on this number). If, for an attributed tree transducer M and for a derivation $s_{in}(e) \Rightarrow^+_{root(e)} t$, the size of t is at least the pumping index n_M, then e has the desired height.

Lemma 3.3 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with k_s synthesized attributes, k_i inherited attributes, and with pumping index n_M. Let $t \in L_{out}(M)$.

If $size(t) \geq n_M$, then for every $e \in T\langle\Sigma\rangle$ such that $t = nf(\Rightarrow_{root(e)}, s_{in}(e))$, the height $height(e) \geq 2^{k_i} \cdot (2^{k_s}-1) + 2$.

Proof. Consider $t \in L_{out}(M)$ with $size(t) \geq n_M$. We examine a control tree $\tilde{e} = root(e)$ with a certain derivation $s_{in}(e) \Rightarrow^+_{\tilde{e}} t$. We abbreviate this derivation by d and the number of derivation steps of d by $length(d)$. The proof consists of a sequence of five implications. First, we list these implications and afterwards we give some explanations.

(1) If $size(t) \geq n_M = 1 + l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$,
then $length(d) \geq 1 + \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$.

(2) If $length(d) \geq 1 + \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$, then $card(att(\tilde{e})) \geq 2 + (k_s + k_i) \cdot n'_M$.

(3) If $card(att(\tilde{e})) \geq 2 + (k_s + k_i) \cdot n'_M$, then $card(att(e)) \geq 1 + (k_s + k_i) \cdot n'_M$.

(4) If $card(att(e)) \geq 1 + (k_s + k_i) \cdot n'_M$, then $size(e) \geq 1 + n'_M$.

(5) If $size(e) \geq 1 + n'_M = 1 + \sum_{j=0}^{2^{k_i}\cdot(2^{k_s}-1)} (m_M)^j$,
then $height(e) \geq 2^{k_i} \cdot (2^{k_s} - 1) + 2$.

(1) Since l_M is the maximum number of output symbols in the right-hand sides of the rules of M, $\sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ rule applications can produce at most $l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ output symbols. Hence, since $size(t) \geq 1 + l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$, it needs at least $1 + \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ rule applications to generate t.

(2) Since every attribute occurrence can call at most c_M other attribute occurrences in one derivation step, $1 + (k_s + k_i) \cdot n'_M$ different attribute occurrences of $\tilde{e}$ can cause at most $\sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ rule applications during the whole derivation d. To understand this fact, we can construct the calling tree of d with attribute occurrences of $\tilde{e}$ as nodes: the root of this tree is labeled $s_{in}(e)$; every node of the tree labeled $a(p)$ has as many sons as there are attribute occurrences in t' with $a(p) \Rightarrow_{\tilde{e}} t'$; the sons are labeled by the different attribute occurrences. It is easy to observe that the length $length(d)$ of the derivation d is equal to the size of the calling tree. Under the assumption that there are at most $1 + (k_s + k_i) \cdot n'_M$ different attribute occurrences of $\tilde{e}$, the height of the calling tree is at most $1 + (k_s + k_i) \cdot n'_M$, because M is noncircular. Thus its size is at most $\sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$. Hence, since $length(d) \geq 1 + \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$, we have at least $2 + (k_s + k_i) \cdot n'_M$ different attribute occurrences of $\tilde{e}$.

(3) At the root of $\tilde{e}$ we only have the attribute occurrence $s_{in}(e)$, thus there exist at least $1 + (k_s + k_i) \cdot n'_M$ attribute occurrences of e.

(4) Since M has $k_s + k_i$ attributes, an input tree e with n'_M nodes can only have $(k_s + k_i) \cdot n'_M$ attribute occurrences. Hence, since $card(att(e)) \geq 1 + (k_s + k_i) \cdot n'_M$, we must have $size(e) \geq 1 + n'_M$.

(5) Since m_M is the maximal rank of the input symbols, an input tree with height $2^{k_i} \cdot (2^{k_s} - 1) + 1$ can only have the size $\sum_{j=0}^{2^{k_i}\cdot(2^{k_s}-1)} (m_M)^j$. Hence, since

$size(e) \geq 1 + n'_M = 1 + \sum_{j=0}^{2^{k_i}\cdot(2^{k_s}-1)} (m_M)^j$, we must have $height(e) \geq 2^{k_i} \cdot (2^{k_s} - 1) + 2$. □

Theorem 3.4 (Pumping Lemma)
Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an si-tree transducer with system $A = (A, A_s, A_i)$ of attributes and pumping index n_M.
For every $t \in L_{out}(M)$ with $size(t) \geq n_M$

- there exist three ranked alphabets
  - $(U_s, rank_{U_s})$ with $U_s \subseteq A_s$, $card(U_s) \geq 1$, and $rank_{U_s}(s) = 0$ for every $s \in U_s$,
  - $(U_i, rank_{U_i})$ with $U_i \subseteq A_i$ and $rank_{U_i}(i) = 0$ for every $i \in U_i$, and
  - $U = U_s \cup U_i$,
- there exists $\hat{t} \in T\langle \Delta \cup U_s \rangle - T\langle \Delta \rangle$ with $size_\Delta(\hat{t}) \geq 1$,
- for every $i \in U_i$, there exists a tree $\hat{t}_i \in T\langle \Delta \cup U_s \rangle$with $size_\Delta(\hat{t}_i) \geq 1$,
- for every $s \in U_s$, there exists a tree $t_s \in T\langle \Delta \cup U \rangle$ with $1 \leq size_\Delta(t_s) < n_M$,
- for every $i \in U_i$, there exists a tree $t_i \in T\langle \Delta \cup U \rangle$ with $1 \leq size_\Delta(t_i) < n_M$,
- for every $s \in U_s$, there exists a tree $\hat{t}_s \in T\langle \Delta \cup U_i \rangle$ with $1 \leq size_\Delta(\hat{t}_s) < n_M$,

with

- for every $s \in U_s$, the symbol s occurs in $\hat{t}$ or there is an $i' \in U_i$ such that s occurs in $\hat{t}_{i'}$,
- for every $s \in U_s$, there is an $s' \in U_s$ such that s occurs in $t_{s'}$ or there is an $i' \in U_i$ such that s occurs in $t_{i'}$,
- for every $i \in U_i$, there is an $s' \in U_s$ such that i occurs in $t_{s'}$ or there is an $i' \in U_i$ such that i occurs in $t_{i'}$,
- for every $i \in U_i$, there is an $s' \in U_s$ such that i occurs in $\hat{t}_{s'}$,

such that $t = \underline{tree}(1)$ and for every $r \geq 0$, the tree $\underline{tree}(r) \in L_{out}(M)$. The function

$\underline{tree} : I\!N \longrightarrow T\langle \Delta \rangle$

is for every $r \geq 0$ defined by $\underline{tree}(r) = \hat{t}\,[s/\underline{tree}'(s, r, 1)\ ;\ s \in U_s]$, where the partial function

$\underline{tree'} : U \times \boldsymbol{N} \times \boldsymbol{N} \to T\langle\Delta\rangle$ is defined as follows:
For every $s \in U_s$ and $r \geq 0$, if $l \in [r]$,
$\underline{tree'}(s,r,l) = t_s[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s,\ i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.
For every $s \in U_s$ and $r \geq 0$, if $l = r+1$,
$\underline{tree'}(s,r,l) = \hat{t}_s[i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.
For every $i \in U_i$ and $r \geq 0$, if $l \in [r]$,
$\underline{tree'}(i,r,l) = t_i[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s,\ i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.
For every $i \in U_i$ and $r \geq 0$, if $l = 0$,
$\underline{tree'}(i,r,l) = \hat{t}_i[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s]$.

Proof. Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an si-tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes, k_s synthesized attributes, and k_i inherited attributes.

Consider $t \in L_{out}(M)$ with $size(t) \geq n_M$. By Lemma 3.3 we know that, for every control tree $\tilde{e} = root(e)$ with $s_{in}(\varepsilon) \Rightarrow_{\tilde{e}}^{+} t$, the condition $height(e) \geq 2^{k_i} \cdot (2^{k_s} - 1) + 2$ holds.

We choose a control tree $\tilde{e} = root(e)$, a derivation $d = (s_{in}(\varepsilon) \Rightarrow_{\tilde{e}}^{+} t)$, and a path p with maximal length from the root of $\tilde{e}$ to a leaf of $\tilde{e}$. Then we know that $|p| \geq 2^{k_i} \cdot (2^{k_s} - 1) + 2 > 2^{k_i} \cdot (2^{k_s} - 1)$. Note that here it would have been sufficient to have $|p| \geq 2^{k_i} \cdot (2^{k_s} - 1) + 1$, but later in the proof of the size conditions for the output patterns we again make use of the pumping index n_M to avoid the definition of a new constant. Otherwise we would have had another formulation of the pumping lemma with two constants (like in [BPS61], there the constants are called p and q).

Since there are exactly 2^{k_i} possibilities to choose an arbitrary subset of the k_i inherited attributes and since there are exactly $2^{k_s} - 1$ possibilities to choose an arbitrary, nonempty subset of the k_s synthesized attributes, we have that

$$card(\{attset(\tilde{e}, p') \mid p' \neq \varepsilon, \text{ and } p' \text{ is a prefix of } p\}) \leq 2^{k_i} \cdot (2^{k_s} - 1).$$

Since $|p| > 2^{k_i} \cdot (2^{k_s} - 1)$, there must exist strings $p_1 \neq \varepsilon$, $p_2 \neq \varepsilon$ and p_3 with $p = p_1 p_2 p_3$, such that

$$attset(\tilde{e}, p_1) = attset(\tilde{e}, p_1 p_2).$$

We choose p_1, p_2, and p_3 such that $|p_2 p_3|$ is minimal. This means that we take the first repetition of $attset(\tilde{e}, p')$, where p' is a prefix of p, beginning from the leaf at p. Then we know that $|p_2 p_3| \leq 2^{k_i} \cdot (2^{k_s} - 1)$, because otherwise there is another repetition of $attset$ in that part of p between $node(\tilde{e}, p_1)$ and $node(\tilde{e}, p_1 p_2 p_3)$, in contradiction to $|p_2 p_3|$ being minimal.

We define the subsets $U_s \subseteq A_s$ and $U_i \subseteq A_i$, such that

$$U_s = attset(\tilde{e}, p_1) \cap A_s \qquad \text{and} \qquad U_i = attset(\tilde{e}, p_1) \cap A_i.$$

In fact, $card(U_s) \geq 1$, because M is visiting and thus every symbol of the control tree must be visited by a synthesized attribute.

Let $w \notin \Sigma_+$ with $rank(w) = 0$. We define trees $e' \in T\langle\Sigma_+ \cup \{w\}\rangle$ and $e'' \in T\langle\Sigma \cup \{w\}\rangle$, where both, e' and e'', have exactly one occurrence of w, and $e''' \in T\langle\Sigma\rangle$ with the help of p_1, p_2 and p_3 as follows:

$$\begin{aligned} e' &= \tilde{e}[p_1 \leftarrow w] \\ e'' &= subtree(\tilde{e}[p_1p_2 \leftarrow w], p_1) \\ e''' &= subtree(\tilde{e}, p_1p_2) \end{aligned}$$

Then the representation $\tilde{e} = e'[w/e''[w/e''']]$ holds. The reader can find these patterns of $\tilde{e}$ and the paths leading to them in Figure 3.

In the sequel we need the sets P_1, P_2, and P_3 of paths, which lead from the root of $\tilde{e}$ to the nodes in the three parts e', e'', and e''', respectively:

$$\begin{aligned} P_1 &= paths(e') - \{p_1\} \\ P_2 &= (\{p_1\} \cdot paths(e'')) - \{p_1p_2\} \\ P_3 &= \{p_1p_2\} \cdot paths(e''') \end{aligned}$$

Note that the path p_1 leading to the root of e'' is excluded from P_1 and that the path p_1p_2 leading to the root of e''' is excluded from P_2.

Now we calculate, roughly speaking, the values of the inside attribute occurrences of the patterns e', e'', and e''' as functions in the values of the outside attribute occurrences of the same patterns in order to gain the desired output patterns that are needed for the pumping process. Therefore we restrict the derivation relation of M to the sets P_1, P_2, and P_3, respectively, as it is defined in Definition 2.6.

For the definition of the output patterns, we use symbols from the ranked alphabets $(U_s, rank_{U_s})$ and $(U_i, rank_{U_i})$ with $rank_{U_s}(s) = 0$ for every $s \in U_s$, with $rank_{U_i}(i) = 0$ for every $i \in U_i$, and with $U = U_s \cup U_i$. We choose exactly the attributes as names for the symbols, to emphasize their strong connection, although they have different ranks. It is easy to decide from the context in which the names occur, whether symbols or attributes are concerned at a time.

Now we can define (cf. Figure 3):

$$\hat{t} = nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))[s'(p_1)/s' \,;\, s' \in U_s].$$

For every $s \in U_s$,

$$\begin{aligned} t_s &= nf(\Rightarrow_{\tilde{e},P_2}, s(p_1))[s'(p_1p_2)/s' \,;\, s' \in U_s,\ i'(p_1)/i' \,;\, i' \in U_i] \quad \text{and} \\ \hat{t}_s &= nf(\Rightarrow_{\tilde{e},P_3}, s(p_1p_2))[i'(p_1p_2)/i' \,;\, i' \in U_i]. \end{aligned}$$

For every $i \in U_i$,

$$\begin{aligned} t_i &= nf(\Rightarrow_{\tilde{e},P_2}, i(p_1p_2))[s'(p_1p_2)/s' \,;\, s' \in U_s,\ i'(p_1)/i' \,;\, i' \in U_i] \quad \text{and} \\ \hat{t}_i &= nf(\Rightarrow_{\tilde{e},P_1}, i(p_1))[s'(p_1)/s' \,;\, s' \in U_s]. \end{aligned}$$

Note that, by the definition of $\Rightarrow_{\tilde{e},P_2}$, the inherited attribute occurrences $i'(p_1)$ cannot be evaluated and thus may occur in $nf(\Rightarrow_{\tilde{e},P_2}, s(p_1))$ and $nf(\Rightarrow_{\tilde{e},P_2}, i(p_1p_2))$. The same holds for $\Rightarrow_{\tilde{e},P_3}$ and the inherited attribute occurrences $i'(p_1p_2)$ that may occur in $nf(\Rightarrow_{\tilde{e},P_3}, s(p_1p_2))$.

By this definition, every pattern has the type, which is required by the pumping lemma. We only have to check that $\hat{t} \notin T\langle\Delta\rangle$. Again the reason is that every symbol of the control tree must be visited by synthesized attributes. Thus, one of the synthesized attribute occurrences $s(p_1)$ must be called directly from $s_{in}(\varepsilon)$

via a sequence of attribute occurrences of e'. Otherwise the derivation would never reach e''.

Now we prove the size conditions for the patterns:

(a) $size_\Delta(\hat{t}) \geq 1$:

We have $size_\Delta(\hat{t}) \geq 1$, because $p_1 \neq \varepsilon$ and thus there must be at least one rule application to calculate $nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(e)) \neq s_{in}(e)$. Note that M is producing.

(b) For every $i \in U_i$, $size_\Delta(\hat{t}_i) \geq 1$:

We have $size_\Delta(\hat{t}_i) \geq 1$, because $nf(\Rightarrow_{\tilde{e},P_1}, i(p_1))$ can only consist of output symbols and attribute occurrences $s'(p_1)$ and thus cannot be equal to $i(p_1)$. Again there is at least one rule application to calculate the normal form.

(c) For every $s \in U_s$, $1 \leq size_\Delta(t_s) < n_M$:

We have $size_\Delta(t_s) \geq 1$, because $p_2 \neq \varepsilon$ and thus there must be at least one rule application to calculate $nf(\Rightarrow_{\tilde{e},P_2}, s(p_1)) \neq s(p_1)$. We have $size_\Delta(t_s) < n_M$, because the calculation of $nf(\Rightarrow_{\tilde{e},P_2}, s(p_1))$ only takes place on the part e'' of the control tree. Since p is a longest path in $\tilde{e}$, its subpath p_2p_3 with $|p_2p_3| \leq 2^{k_i} \cdot (2^{k_s} - 1)$ is a longest path of $e''[w/e''']$ and thus e'' can have no path with a length greater than $2^{k_i} \cdot (2^{k_s} - 1)$. Then $height(e'') \leq 2^{k_i} \cdot (2^{k_s} - 1) + 1$ and (with a reverse argumentation to fix $height(e)$ in the proof of Lemma 3.3 we get $size(e'') \leq \sum_{j=0}^{2^{k_i} \cdot (2^{k_s}-1)} (m_M)^j = n'_M$, we have less than $1 + (k_s + k_i) \cdot n'_M$ attribute occurrences of e'', we have less than $\sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ rule applications to generate t_s and thus $size_\Delta(t_s) < l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j < n_M$.

(d) For every $i \in U_i$, $1 \leq size_\Delta(t_i) < n_M$:

We have $size_\Delta(t_i) \geq 1$, because $p_2 \neq \varepsilon$ and thus there must be at least one rule application to calculate $nf(\Rightarrow_{\tilde{e},P_2}, i(p_1p_2)) \neq i(p_1p_2)$. The proof for $size_\Delta(t_i) < n_M$ is analogous to that in (c).

(e) For every $s \in U_s$, $1 \leq size_\Delta(\hat{t}_s) < n_M$:

We have $size_\Delta(\hat{t}_s) \geq 1$, because $nf(\Rightarrow_{\tilde{e},P_3}, s(p_1p_2))$ can only consist of output symbols and attribute occurrences $i'(p_1p_2)$ and thus cannot be equal to $s(p_1p_2)$. Again there is at least one rule application to calculate the normal form. We have $size_\Delta(\hat{t}_s) < n_M$, because the calculation of $nf(\Rightarrow_{\tilde{e},P_3}, s(p_1p_2))$ only takes place on the part e''' of the control tree. Since p is a longest path in $\tilde{e}$, its subpath p_3 with $|p_3| < |p_2p_3| \leq 2^{k_i} \cdot (2^{k_s} - 1)$ is a longest path of e'''. Now we can apply the same argumentation as in (c).

In the next step we have to check, whether the symbols $s \in U_s$ and $i \in U_i$ occur at least once in the desired patterns of t. We show the proof only for the occurrences of $s \in U_s$ in the tree $\hat{t}$ or in a tree $\hat{t}_{i'}$ for some $i' \in U_i$. The other cases can be treated analogous. The proof works by contradiction:

If there is an $s \in U_s$ such that s does not occur in $\hat{t}$ and not in $\hat{t}_{i'}$ for every $i' \in U_i$, then, by the definition of the patterns of t, the attribute occurrence $s(p_1)$ does not occur in $nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))$ and not in $nf(\Rightarrow_{\tilde{e},P_1}, i'(p_1))$ for every $i' \in U_i$. But the calculation of these normal forms are the only parts of the derivation d, in which the attribute occurrence $s(p_1)$ can be introduced into the derivation. Thus $s(p_1)$ cannot occur in d, in contradiction to $s \in attset(d, p_1) = attset(\tilde{e}, p_1)$.

The last item of this proof is to show that $t = \underline{tree}(1)$ and that for every $r \geq 0$, the property $\underline{tree}(r) \in L_{out}(M)$ holds.

We abbreviate the control tree, which is built up by repeating $r \geq 0$ times the pattern e'', by $\tilde{e}_r$:

$$\tilde{e}_r = e'[w/\underbrace{e''[w/\ldots e''[w/}_{r \text{ times}} e''' \underbrace{]\ldots]}_{r \text{ times}}]$$

Thus, in particular, $\tilde{e} = \tilde{e}_1$.

First we have to verify the following Statements (1a) and (1b) concerning the function $\underline{tree'}$:

(1a) For every $s \in U_s$, $r \geq 0$, $1 \leq l \leq r+1$, $\underline{tree'}(s, r, l) = nf(\Rightarrow_{\tilde{e}_r}, s(p_1 p_2^{l-1}))$.
(1b) For every $i \in U_i$, $r \geq 0$, $0 \leq l \leq r$, $\underline{tree'}(i, r, l) = nf(\Rightarrow_{\tilde{e}_r}, i(p_1 p_2^l))$.

Since M is noncircular, there must exist an order in which, for every $r \geq 0$, the attribute occurrences of the set $\{s(p_1 p_2^l) \mid s \in U_s, 0 \leq l \leq r\} \cup \{i(p_1 p_2^l) \mid i \in U_i, 0 \leq l \leq r\}$ can be evaluated. This order induces an order θ on the set $\{\underline{tree'}(s, r, l) \mid s \in U_s, 1 \leq l \leq r+1\} \cup \{\underline{tree'}(i, r, l) \mid i \in U_i, 0 \leq l \leq r\}$ of function calls and thus it is guaranteed that the recursive function $\underline{tree'}$ is well defined.

If, for example, the evaluation of $\underline{tree'}(s, r, l)$ forces us to evaluate $\underline{tree'}(s', r, l+1)$, then, for every $1 \leq l \leq r$, the attribute occurrence $s'(p_1 p_2^l)$ has to appear earlier than the attribute occurrence $s(p_1 p_2^{l-1})$ in an order of the above attribute occurrences. But this is guaranteed, because in this case t_s must contain a symbol s' (compare the definition of $\underline{tree'}$ in Theorem 3.4) and by the definition of t_s, we must have an attribute occurrence $s'(p_1 p_2)$ in $nf(\Rightarrow_{\tilde{e},P_2}, s(p_1))$. Hence, $s'(p_1 p_2)$ must be evaluated before $s(p_1)$ and thus, for every $1 \leq l \leq r$, $s'(p_1 p_2^l)$ must be evaluated before $s(p_1 p_2^{l-1})$.

Now we take an arbitrary such order θ of function calls which can be considered as a string of length $(r+1) \cdot card(U)$. Then we can prove the Statements (1a) and (1b) by finite (mathematical) induction on ν with $1 \leq \nu \leq (r+1) \cdot card(U)$, i.e., ν is a position in this string. Depending on the function call at position ν, we have to prove either the statement $\underline{tree'}(s, r, l) = nf(\Rightarrow_{\tilde{e}_r}, s(p_1 p_2^{l-1}))$ (if the ν–th function call is $\underline{tree'}(s, r, l)$) or the statement $\underline{tree'}(i, r, l) = nf(\Rightarrow_{\tilde{e}_r}, i(p_1 p_2^l))$ (if the ν–th function call is $\underline{tree'}(i, r, l)$). If we want to prove the statement for the function call at position ν in θ, then we can use the induction hypothesis which says that, for every function call at position ν' with $1 \leq \nu' < \nu$, the corresponding statement holds.

Case (a): The function call at position ν is $\underline{tree'}(s, r, l)$ with $s \in U_s$, $r \geq 0$, and $1 \leq l \leq r+1$. Thus we have to prove the statement $\underline{tree'}(s, r, l) = nf(\Rightarrow_{\tilde{e}_r}, s(p_1 p_2^{l-1}))$. There are two cases:

Case I: $1 \leq l \leq r$

$$
\begin{aligned}
& \underline{tree'}(s, r, l) \\
= \; & t_s[s'/\underline{tree'}(s', r, l+1) \; ; \; s' \in U_s, \; i'/\underline{tree'}(i', r, l-1) \; ; \; i' \in U_i] \\
& \qquad \text{(Definition of } \underline{tree'}\text{)} \\
= \; & t_s[s'/nf(\Rightarrow_{\bar{z}_r}, s'(p_1 p_2^l)) \; ; \; s' \in U_s, \; i'/nf(\Rightarrow_{\bar{z}_r}, i'(p_1 p_2^{l-1})) \; ; \; i' \in U_i] \\
& \qquad \text{(Induction Hypothesis for function} \\
& \qquad \text{calls with positions less than } \nu\text{)} \\
= \; & nf(\Rightarrow_{\bar{z}_r}, t_s[s'/s'(p_1 p_2^l) \; ; \; s' \in U_s, \; i'/i'(p_1 p_2^{l-1}) \; ; \; i' \in U_i]) \\
= \; & nf(\Rightarrow_{\bar{z}_r}, s(p_1 p_2^{l-1})) \qquad \text{(Calculation on the } l\text{-th occurrence of } e''\text{)}
\end{aligned}
$$

Case II: $l = r + 1$

$$
\begin{aligned}
& \underline{tree'}(s, r, r+1) \\
= \; & \hat{t}_s[i'/\underline{tree'}(i', r, r) \; ; \; i' \in U_i] \qquad \text{(Definition of } \underline{tree'}\text{)} \\
= \; & \hat{t}_s[i'/nf(\Rightarrow_{\bar{z}_r}, i'(p_1 p_2^r)) \; ; \; i' \in U_i] \qquad \text{(Induction Hypothesis for function} \\
& \qquad \text{calls with positions less than } \nu\text{)} \\
= \; & nf(\Rightarrow_{\bar{z}_r}, \hat{t}_s[i'/i'(p_1 p_2^r) \; ; \; i' \in U_i]) \\
= \; & nf(\Rightarrow_{\bar{z}_r}, s(p_1 p_2^r)) \qquad \text{(Calculation on } e'''\text{)}
\end{aligned}
$$

Case (b): The function call at position ν is $\underline{tree'}(i, r, l)$ with $i \in U_i$, $r \geq 0$, and $0 \leq l \leq r$. Thus we have to prove the statement $\underline{tree'}(i, r, l) = nf(\Rightarrow_{\bar{z}_r}, i(p_1 p_2^l))$. There are two cases:

Case I: $1 \leq l \leq r$

$$
\begin{aligned}
& \underline{tree'}(i, r, l) \\
= \; & t_i[s'/\underline{tree'}(s', r, l+1) \; ; \; s' \in U_s, \; i'/\underline{tree'}(i', r, l-1) \; ; \; i' \in U_i] \\
& \qquad \text{(Definition of } \underline{tree'}\text{)} \\
= \; & t_i[s'/nf(\Rightarrow_{\bar{z}_r}, s'(p_1 p_2^l)) \; ; \; s' \in U_s, \; i'/nf(\Rightarrow_{\bar{z}_r}, i'(p_1 p_2^{l-1})) \; ; \; i' \in U_i] \\
& \qquad \text{(Induction Hypothesis for function} \\
& \qquad \text{calls with positions less than } \nu\text{)} \\
= \; & nf(\Rightarrow_{\bar{z}_r}, t_i[s'/s'(p_1 p_2^l) \; ; \; s' \in U_s, \; i'/i'(p_1 p_2^{l-1}) \; ; \; i' \in U_i]) \\
= \; & nf(\Rightarrow_{\bar{z}_r}, i(p_1 p_2^l)) \qquad \text{(Calculation on the } l\text{-th occurrence of } e''\text{)}
\end{aligned}
$$

Case II: $l = 0$

$$
\begin{aligned}
& \underline{tree'}(i, r, 0) \\
= \; & \hat{t}_i[s'/\underline{tree'}(s', r, 1) \; ; \; s' \in U_s] \qquad \text{(Definition of } \underline{tree'}\text{)} \\
= \; & \hat{t}_i[s'/nf(\Rightarrow_{\bar{z}_r}, s'(p_1)) \; ; \; s' \in U_s] \qquad \text{(Induction Hypothesis for function} \\
& \qquad \text{calls with positions less than } \nu\text{)} \\
= \; & nf(\Rightarrow_{\bar{z}_r}, \hat{t}_i[s'/s'(p_1) \; ; \; s' \in U_s]) \\
= \; & nf(\Rightarrow_{\bar{z}_r}, i(p_1)) \qquad \text{(Calculation on } e'\text{)}
\end{aligned}
$$

Then we can prove for every $r \geq 0$ the equation $\underline{tree}(r) = nf(\Rightarrow_{\bar{z}_r}, s_{in}(e))$:

$$
\begin{aligned}
& \underline{tree}(r) \\
= \; & \hat{t}\,[s/\underline{tree}'(s,r,1)\ ;\ s \in U_s] && \text{(Definition of } \underline{tree}\text{)} \\
= \; & nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))[s(p_1)/s\ ;\ s \in U_s][s/\underline{tree}'(s,r,1)\ ;\ s \in U_s] \\
& && \text{(Definition of } \hat{t}\text{)} \\
= \; & nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))[s(p_1)/\underline{tree}'(s,r,1)\ ;\ s \in U_s] \\
= \; & nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))[s(p_1)/nf(\Rightarrow_{\tilde{e}_r}, s(p_1))\ ;\ s \in U_s] && \text{(Statement (1a))} \\
= \; & nf(\Rightarrow_{\tilde{e}_r,P_1}, s_{in}(\varepsilon))[s(p_1)/nf(\Rightarrow_{\tilde{e}_r}, s(p_1))\ ;\ s \in U_s] && \text{(Subterm } e' \text{ in } \tilde{e}_r \\
& && \text{unchanged)} \\
= \; & nf(\Rightarrow_{\tilde{e}_r}, s_{in}(\varepsilon))
\end{aligned}
$$

This equation has the two desired consequences that finish the proof of the pumping lemma:

- $\underline{tree}(1) = nf(\Rightarrow_{\tilde{e}_1}, s_{in}(\varepsilon)) = nf(\Rightarrow_{\tilde{e}}, s_{in}(\varepsilon)) = t.$
- For every $r \geq 0$, $\underline{tree}(r) = nf(\Rightarrow_{\tilde{e}_r}, s_{in}(\varepsilon)) \in L_{out}(M)$, because $\tau(M)(e'_r) = \underline{tree}(r)$ where $\tilde{e}_r = root(e'_r)$. □

We want to conclude this section with an observation concerning the requirements of the attributed tree transducers to be producing and visiting.

If we had dropped the "producing–condition", then the pumping process itself would not have been affected. But it would have been impossible to prove that the output patterns consist of at least one output symbol. In the next section we shall see that the applications of the pumping lemma demonstrated there, are no more feasible without this size–condition.

If we had dropped the "visiting–condition", then the proof of the pumping lemma itself would have been impossible. Since for the control tree $\tilde{e}$ and for every subpath p' of the chosen path p, $attset(\tilde{e}, p') \neq \emptyset$ cannot be guaranteed, the following construction is no more feasible.

4 Applications

Our pumping lemma is usable for the output language of every noncircular, producing, and visiting attributed tree transducer. But, if we take output languages which are constructed over an arbitrary output alphabet, then the application of the pumping lemma is very difficult. Hence we apply our pumping lemma only to output languages with monadic trees.

The following Theorem 4.1 is a specialised version of our pumping lemma for the case of monadic output languages. Observation 4.2 makes a statement about the number of occurrences of the output patterns in the trees $\underline{tree}(0)$ and $\underline{tree}(1)$ in the case of monadic output languages. We use this theorem and this observation in the following proofs.

4.1 Pumping lemma for monadic output languages

In order to simplify the study of this paper we state here a complete monadic version of the pumping lemma instead of giving only the additional conditions.

Theorem 4.1 Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes, pumping index n_M, and $\Delta = \Delta^{(1)} \cup \Delta^{(0)}$. For every $t \in L_{out}(M)$ with $size(t) \geq n_M$

- there exist three ranked alphabets
  - $(U_s, rank_{U_s})$ with $U_s \subseteq A_s$, $card(U_s) \geq 1$, and $rank_{U_s}(s) = 0$ for every $s \in U_s$,
  - $(U_i, rank_{U_i})$ with $U_i \subseteq A_i$ and $rank_{U_i}(i) = 0$ for every $i \in U_i$, and
  - $U = U_s \cup U_i$,
  - with $card(U_s) = card(U_i)$ or $card(U_s) = card(U_i) + 1$,
- there exist $u \in U_s$ and $\hat{t} \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $size_\Delta(\hat{t}) \geq 1$,
- for every $i \in U_i$, there exist $u \in \Delta^{(0)} \cup U_s$ and $\hat{t}_i \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $size_\Delta(\hat{t}_i) \geq 1$,
- for every $s \in U_s$, there exist $u \in U$ and $t_s \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $1 \leq size_\Delta(t_s) < n_M$,
- for every $i \in U_i$, there exist $u \in U$ and $t_i \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $1 \leq size_\Delta(t_i) < n_M$,
- for every $s \in U_s$, there exist $u \in \Delta^{(0)} \cup U_i$ and $\hat{t}_s \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $1 \leq size_\Delta(\hat{t}_s) < n_M$,

such that

- exactly one tree of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ is of type $T\langle\Delta\rangle$, such that
  if $card(U_s) = card(U_i)$, then there is exactly one $i \in U_i$ such that $\hat{t}_i \in T\langle\Delta\rangle$,
  if $card(U_s) = card(U_i) + 1$, then there is exactly one $s \in U_s$ such that $\hat{t}_s \in T\langle\Delta\rangle$,
- for every $s \in U_s$, the symbol s occurs in exactly one tree of the set $\{\hat{t}\} \cup \{\hat{t}_{i'} \mid i' \in U_i\}$,
- for every $s \in U_s$, the symbol s occurs in exactly one tree of the set $\{t_{s'} \mid s' \in U_s\} \cup \{t_{i'} \mid i' \in U_i\}$,
- for every $i \in U_i$, the symbol i occurs in exactly one tree of the set $\{t_{s'} \mid s' \in U_s\} \cup \{t_{i'} \mid i' \in U_i\}$,
- for every $i \in U_i$, the symbol i occurs in exactly one tree of the set $\{\hat{t}_{s'} \mid s' \in U_s\}$,

such that $t = \underline{tree}(1)$ and for every $r \geq 0$, the tree $\underline{tree}(r) \in L_{out}(M)$. The function

$\underline{tree} : \mathbb{N} \longrightarrow T\langle\Delta\rangle$

is for every $r \geq 0$ defined by $\underline{tree}(r) = \hat{t}\,[s/\underline{tree'}(s,r,1)\ ;\ s \in U_s]$, where the partial function

$\underline{tree'} : U \times \mathbb{N} \times \mathbb{N} - \!\!\rightarrow T\langle\Delta\rangle$ is defined as follows:

For every $s \in U_s$ and $r \geq 0$, if $l \in [r]$,

$\underline{tree'}(s,r,l) = t_s[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s,\ i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.

For every $s \in U_s$ and $r \geq 0$, if $l = r+1$,

$\underline{tree'}(s,r,l) = \hat{t}_s[i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.

For every $i \in U_i$ and $r \geq 0$, if $l \in [r]$,

$\underline{tree'}(i,r,l) = t_i[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s,\ i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.

For every $i \in U_i$ and $r \geq 0$, if $l = 0$,

$\underline{tree'}(i,r,l) = \hat{t}_i[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s]$.

Proof. We only have to prove the additional conditions of the pumping lemma. The proof is based on the proof of Theorem 3.4. Thus we make use of some notions which were introduced there.

We first prove the correctness of the substitutions of "occurs in a tree" in Theorem 3.4 by "occurs in exactly one tree". We show the proof only for the occurrence of s in the tree $\hat{t}$ or in a tree $\hat{t}_{i'}$. The other cases can be treated analogous. The proof works by contradiction:

Assume that there is an $s \in U_s$ such that s occurs in at least two different trees of the set $\{\hat{t}\} \cup \{\hat{t}_{i'} \mid i' \in U_i\}$. Then, by the definition of the patterns of t, the attribute occurrence $s(p_1)$ occurs in two different normal forms of $nf(\Rightarrow_{\hat{\varepsilon},P_1}, s_{in}(\varepsilon))$ and $nf(\Rightarrow_{\hat{\varepsilon},P_1}, i'(p_1))$ for $i' \in U_i$. The calculation of these normal forms correspond to different parts of the derivation $s_{in}(\varepsilon) \Rightarrow_{\hat{\varepsilon}}^{+} t$. Thus $s(p_1)$ occurs in two different sentential forms of the derivation $s_{in}(\varepsilon) \Rightarrow_{\hat{\varepsilon}}^{+} t$. There must exist $t_1, t_2 \in (\Delta^{(1)})^+$ with $s_{in}(\varepsilon) \Rightarrow_{\hat{\varepsilon}}^{+} t_1 s(p_1) \Rightarrow_{\hat{\varepsilon}}^{+} t_1 t_2 s(p_1) \Rightarrow_{\hat{\varepsilon}}^{+} t$. Consequently, M is circular, which is a contradiction. The conditions that

- there exist $u \in U_s$ and $\hat{t} \in T\langle\Delta^{(1)} \cup \{u\}\rangle$,
- for every $i \in U_i$, there exist $u \in \Delta^{(0)} \cup U_s$ and $\hat{t}_i \in T\langle\Delta^{(1)} \cup \{u\}\rangle$, and
- for every $s \in U_s$, there exist $u \in \Delta^{(0)} \cup U_i$ and $\hat{t}_s \in T\langle\Delta^{(1)} \cup \{u\}\rangle$

are direct consequences of the pumping lemma, because Δ is monadic.

- For every $s \in U_s$, there exist $u \in U$ and $t_s \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ and
- for every $i \in U_i$, there exist $u \in U$ and $t_i \in T\langle\Delta^{(1)} \cup \{u\}\rangle$,

because each of the $card(U)$ symbols of U occurs in exactly one of the $card(U)$ trees of the set $\{t_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\}$, and because each of these trees can contain at most one (and thus exactly one) of the symbols.

We know that each of the $card(U_i)$ symbols of U_i occurs in exactly one of the $card(U_s)$ trees of the set $\{\hat{t}_s \mid s \in U_s\}$, and that each of these trees can contain at most one of the symbols. Thus we must have $card(U_s) \geq card(U_i)$. We also know that each of the $card(U_s)$ symbols of U_s occurs in exactly one of the $card(U_i) + 1$ trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\}$, and that $\hat{t}$ contains exactly one and each of the other trees can contain at most one of the symbols. Thus we must have $card(U_i) \geq card(U_s) - 1$. We can conclude that $card(U_s) = card(U_i)$ or $card(U_s) = card(U_i) + 1$ holds.

If $card(U_s) = card(U_i)$, then every tree $\hat{t}_s$ contains exactly one of the symbols of U_i and every tree $\hat{t}_i$ except one of them contains exactly one of the symbols of U_s. Thus there is exactly one $i \in U_i$ with $\hat{t}_i \in T\langle\Delta\rangle$.

If $card(U_s) = card(U_i)+1$, then every tree $\hat{t}_i$ contains exactly one of the symbols of U_s and every tree $\hat{t}_s$ except one of them contains exactly one of the symbols of U_i. Thus there is exactly one $s \in U_s$ with $\hat{t}_s \in T\langle\Delta\rangle$. □

Observation 4.2 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*-tree transducer with system $A = (A, A_s, A_i)$ of attributes and $\Delta = \Delta^{(1)} \cup \Delta^{(0)}$. Then in Theorem 4.1,

1. $\underline{tree}(0)$ is built up, using each of the trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ exactly once and
2. $t = \underline{tree}(1)$ is built up, using each of the trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ exactly once.

Proof. Again we make use of some notions which were introduced in the proof of Theorem 3.4.

(a) The tree $\hat{t}$ is used exactly once in $\underline{tree}(0)$ and $\underline{tree}(1)$, because $\hat{t}$ is introduced calling the function $\underline{tree'}$ the first time and nowhere else.

(b) The argumentation for the statement that the trees of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ are used at most once in $\underline{tree}(0)$ works as follows by contradiction:

W.l.o.g. we assume that a tree $\hat{t}_i$ is used twice (or more than twice). Then the calculation of $nf(\Rightarrow_{\hat{\varepsilon},P_1}, i(p_1))$ corresponds to different parts of the derivation $s_{in}(\varepsilon) \Rightarrow^+_{\hat{\varepsilon}_0} \underline{tree}(0)$. Thus $i(p_1)$ occurs in different sentential forms of the derivation $s_{in}(\varepsilon) \Rightarrow^+_{\hat{\varepsilon}_0} \underline{tree}(0)$. There must exist $t_1, t_2 \in (\Delta^{(1)})^+$ with $s_{in}(\varepsilon) \Rightarrow^+_{\hat{\varepsilon}_0} t_1 i(p_1) \Rightarrow^+_{\hat{\varepsilon}_0} t_1 t_2 i(p_1) \Rightarrow^+_{\hat{\varepsilon}_0} \underline{tree}(0)$. Consequently, M is circular, which is a contradiction.

(c) The same argumentation can be applied for the proof of the statement that the trees of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ are used at most once in $\underline{tree}(1)$.

(d) The argumentation for the statement that the trees of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ are used at least once in $\underline{tree}(0)$ works as follows by contradiction:

By Theorem 4.1 we have $card(U_s) = card(U_i)$ or $card(U_s) = card(U_i) + 1$. We show the proof only for the case $card(U_s) = card(U_i)$. The other case can be proved analogous.

We let $k = card(U_s) = card(U_i)$, $U_s = \{s_1, \ldots, s_k\}$, and $U_i = \{i_1, \ldots, i_k\}$.

Assume that not all of the desired output patterns occur in $\underline{tree}(0)$. The number of used trees $\hat{t}_i$ with $i \in U_i$ and the number of used trees $\hat{t}_s$ with $s \in U_s$ has to be equal, because the process of building up $\underline{tree}(0)$ starts with $\hat{t}$, it must end with the only tree $\hat{t}_i \in T\langle\Delta\rangle$ by Theorem 4.1, and the use of trees $\hat{t}_i$ with $i \in U_i$ and of trees $\hat{t}_s$ with $s \in U_s$ must alternate, as can be seen observing the function $\underline{tree'}$.

Thus we can assume that there is a $k' \in [k-1]$, such that only the patterns $\hat{t}_{i_1}, \ldots, \hat{t}_{i_{k'}}, \hat{t}_{s_1}, \ldots, \hat{t}_{s_{k'}}$ occur in $\underline{tree}(0)$ (possibly by renaming the trees). We construct a circularity in $\tilde{e}_0$ with the remaining patterns $\hat{t}_{i_{k'+1}}, \ldots, \hat{t}_{i_k}, \hat{t}_{s_{k'+1}}, \ldots, \hat{t}_{s_k}$ which can not be of type $T\langle\Delta\rangle$, as follows:

Because of Theorem 4.1 and because the symbols $s_1, \ldots, s_{k'}, i_1, \ldots, i_{k'}$ must occur in the patterns which are used to construct $\underline{tree}(0)$, we know:

- For every j with $k'+1 \leq j \leq k$, the tree $\hat{t}_{i_j} \in T\langle\Delta^{(1)} \cup \{s_{k'+1}, \ldots, s_k\}\rangle$,
- for every j with $k'+1 \leq j \leq k$, the tree $\hat{t}_{s_j} \in T\langle\Delta^{(1)} \cup \{i_{k'+1}, \ldots, i_k\}\rangle$,
- and every symbol $s_{k'+1}, \ldots, s_k, i_{k'+1}, \ldots, i_k$ must occur in exactly one tree of the set $\{\hat{t}_{i_{k'+1}}, \ldots, \hat{t}_{i_k}, \hat{t}_{s_{k'+1}}, \ldots, \hat{t}_{s_k}\}$.

Thus, possibly by renaming the trees, there must exist $k'' \in [k-k']$ with:

- For every j with $k'+1 \leq j \leq k'+k''$, the tree $\hat{t}_{i_j} \in T\langle\Delta^{(1)} \cup \{s_j\}\rangle$,
- for every j with $k'+1 \leq j \leq k'+k''-1$, the tree $\hat{t}_{s_j} \in T\langle\Delta^{(1)} \cup \{i_{j+1}\}\rangle$,
- and $\hat{t}_{s_{k'+k''}} \in T\langle\Delta^{(1)} \cup \{i_{k'+1}\}\rangle$.

By the definition of the patterns in the proof of Theorem 3.4 we know that these patterns correspond to normal forms of certain attribute occurrences and we can construct a derivation on the control tree $\tilde{e}_0$ as follows:

$$
\begin{array}{ll}
 & i_{k'+1}(p_1) \\
\Rightarrow^+_{\tilde{e}_0} & \hat{t}_{i_{k'+1}} s_{k'+1}(p_1) \\
\Rightarrow^+_{\tilde{e}_0} & \hat{t}_{i_{k'+1}} \hat{t}_{s_{k'+1}} i_{k'+2}(p_1) \\
\vdots & \\
\Rightarrow^+_{\tilde{e}_0} & \hat{t}_{i_{k'+1}} \hat{t}_{s_{k'+1}} \ldots \hat{t}_{i_{k'+k''}} s_{k'+k''}(p_1) \\
\Rightarrow^+_{\tilde{e}_0} & \hat{t}_{i_{k'+1}} \hat{t}_{s_{k'+1}} \ldots \hat{t}_{i_{k'+k''}} \hat{t}_{s_{k'+k''}} i_{k'+1}(p_1)
\end{array}
$$

We can conclude that M is circular, which is a contradiction. An example situation which would be a consequence of the assumption that not all of the desired output patterns occur in $\underline{tree}(0)$ is shown in Figure 10.

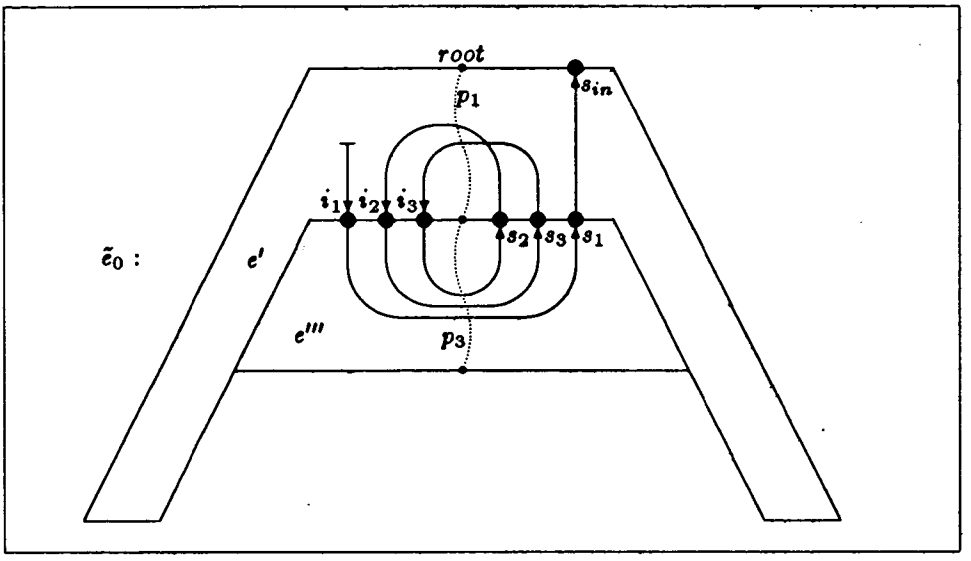


Figure 10: Circularity in $\tilde{e}_0$ with $k = 3$, $k' = 1$ and $k'' = 2$.

(e) The trees of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ are used at least once in $\underline{tree}(1)$, because these patterns correspond to parts of the derivation $s_{in}(e) \Rightarrow_{\tilde{e}}^{+} t = \underline{tree}(1)$ by the definition of the output patterns in the proof of Theorem 3.4. □

4.2 Arithmetic Proof

It is known from Lemma 4.1 of [Fül81] that, if M is an attributed tree transducer and if $\tau(M)(e) = t$ for an input tree e and an output tree t, then there is a constant $c > 0$ such that $height(t) \leq c \cdot size(e)$ holds. Thus, there cannot exist an attributed tree transducer M, which calculates the tree transformation $\tau(M) : T\langle\{\gamma^{(1)}, \alpha^{(0)}\}\rangle \longrightarrow T\langle\{B^{(1)}, E^{(0)}\}\rangle$ with $\tau(M)(\gamma^n\alpha) = B^{2^n}E$ for every $n \geq 0$. We only mention here that there is a macro tree transducer (cf. Example 4.3 of [EV85]) which calculates this tree transformation.

If we do not restrict the input trees to be monadic trees, then the lemma of Fülöp says nothing about whether an attributed tree transducer M' exists computing the tree transformation $\tau(M') : T\langle\Sigma\rangle \longrightarrow T\langle\{B^{(1)}, E^{(0)}\}\rangle$ with $L_{out}(M') = \{B^{2^n}E \mid n \geq 0\}$. Such a producing and visiting attributed tree transducer cannot exist, because we can use our pumping lemma to prove that $\{B^{2^n}E \mid n \geq 0\} \notin SIT_{out}$ holds.

We call the following kind of proof arithmetic proof, because we use arithmetic arguments while applying the pumping lemma.

Theorem 4.3 $\{B^{2^n}E \mid n \geq 0\} \notin SIT_{out}$

Proof. Assume that there is an *si*-tree transducer $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ with system $\mathcal{A} = (A, A_s, A_i)$ of attributes and $L_{out}(M) = \{B^{2^n}E \mid n \geq 0\}$. By Theorem 4.1, for every $t \in L_{out}(M)$ with $size(t) \geq n_M$, where $n_M \geq 1$ is the pumping index of M, certain properties hold. Consider $t = B^{2^{n_M \cdot card(A)}}E$; clearly, $size(t) \geq n_M$.

According to Theorem 4.1 there exist $U_s \subseteq A_s$ with $card(U_s) \geq 1$, $U_i \subseteq A_i$, a tree $\hat{t}$, trees $\hat{t}_i, t_i$ for every $i \in U_i$, and trees $\hat{t}_s, t_s$ for every $s \in U_s$ fulfilling the conditions of Theorem 4.1, such that $t = \underline{tree}(1)$.

$t = \underline{tree}(1)$ is built up, using each of the trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ exactly once, because of Observation 4.2. $\underline{tree}(0)$ is built up, using each of the trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ exactly once, because of Observation 4.2.

Thus we can estimate $size(\underline{tree}(0))$ with the size conditions of Theorem 4.1 as follows:

$$
\begin{aligned}
& size(\underline{tree}(0)) \\
= \;& size(\underline{tree}(1)) - \textstyle\sum_{s \in U_s} size_\Delta(t_s) - \sum_{i \in U_i} size_\Delta(t_i) \\
\geq \;& 2^{n_M \cdot card(A)} + 1 - (n_M - 1) \cdot (card(U_s) + card(U_i)) && (size_\Delta(t_s) \leq n_M - 1, \\
& && \; size_\Delta(t_i) \leq n_M - 1) \\
\geq \;& 2^{n_M \cdot card(A)} + 1 - (n_M - 1) \cdot (card(A_s) + card(A_i)) \\
= \;& 2^{n_M \cdot card(A)} + 1 - (n_M - 1) \cdot card(A) \\
\geq \;& 2^{n_M \cdot card(A)} + 1 - (n_M \cdot card(A) - 1) \\
> \;& 2^{n_M \cdot card(A)} + 1 - 2^{n_M \cdot card(A) - 1} \\
= \;& 2^{n_M \cdot card(A) - 1} + 1, \qquad \text{and}
\end{aligned}
$$

$$
\begin{aligned}
& size(\underline{tree}(0)) \\
= \;& size(\underline{tree}(1)) - \textstyle\sum_{s \in U_s} size_\Delta(t_s) - \sum_{i \in U_i} size_\Delta(t_i) \\
\leq \;& 2^{n_M \cdot card(A)} + 1 - (card(U_s) + card(U_i)) && (size_\Delta(t_s) \geq 1, \\
& && \; size_\Delta(t_i) \geq 1) \\
< \;& 2^{n_M \cdot card(A)} + 1.
\end{aligned}
$$

Note that the requirement of M to be producing is necessary for this part of the proof.

Thus $2^{n_M \cdot card(A) - 1} + 1 < size(\underline{tree}(0)) < 2^{n_M \cdot card(A)} + 1$ and therefore $\underline{tree}(0) \notin L_{out}(M) = \{B^{2^n}E \mid n \geq 0\}$, contradicting the assumption. □

4.3 Structural Proof

In contrast to the (easier) arithmetic proofs, we want to demonstrate here, how structural properties of a certain output language can be used while applying the pumping lemma for attributed tree transducers. We use the results of this subsection to present a hierarchy for attributed tree transducers with bounded number of attributes.

Lemma 4.4 For every $k \geq 1$, $\{(BD^n)^{2k+1}E \mid n \geq 0\} \notin S_{(k)}I_{(k)}T_{out}$.

Proof. Let $k \geq 1$. Assume that there is an *si*– tree transducer $M = (A, \Delta, \Sigma, s_{in}, root, R)$ with system $A = (A, A_s, A_i)$ of attributes, $L_{out}(M) = \{(BD^n)^{2k+1}E \mid n \geq 0\}$ and with k synthesised attributes and k inherited attributes. By Theorem 4.1, for every $t \in L_{out}(M)$ with $size(t) \geq n_M$, where $n_M \geq 1$ is the pumping index of M, certain properties hold. Consider $t = (BD^{n_M \cdot (2k+1)})^{2k+1}E$; clearly $size(t) \geq n_M$.

According to Theorem 4.1 there exist $U_s \subseteq A_s$ with $card(U_s) \geq 1$, $U_i \subseteq A_i$ with $card(U_s) = card(U_i)$ or $card(U_s) = card(U_i) + 1$. Additionally, there exist a pattern $\hat{t}$, patterns $\hat{t}_i, t_i$ for every $i \in U_i$, and patterns $\hat{t}_s, t_s$ for every $s \in U_s$ fulfilling the conditions of Theorem 4.1, such that $t = \underline{tree}(1)$.

$t = \underline{tree}(1)$ is built up, using each of the patterns of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ exactly once, because of Observation 4.2. In the following, we simply identify these patterns with the sequence of their output symbols from the root to the leaf by dropping the symbols $s \in U_s$ and $i \in U_i$. This notation is slightly inaccurate, but easier to read. We let $k_1 = card(U_s)$, $k_2 = card(U_i)$, $U_s = \{s_1, \ldots, s_{k_1}\}$, and $U_i = \{i_1, \ldots, i_{k_2}\}$.

Case 1: $k_1 = k_2$

In this case we can represent t as follows, where for every $l \in [k_1]$, $t^{(l)}$ is a sequence of patterns taken from the $3k_1$ patterns $t_{s_1}, \ldots, t_{s_{k_1}}, t_{i_1}, \ldots, t_{i_{k_1}}, \hat{t}_{s_1}, \ldots, \hat{t}_{s_{k_1}}$:

$$t = \underline{tree}(1) = \hat{t}\, t^{(1)}\, \hat{t}_{i_1}\, t^{(2)}\, \hat{t}_{i_2} \ldots t^{(k_1)}\, \hat{t}_{i_{k_1}}$$

For every $l \in [k_1]$, the tree $t^{(l)}$ is built up from at least one pattern. It is constructed from at most $2k_1 + 1$ patterns, if the other trees $t^{(l')}$ are built up from exactly one pattern, because each pattern can only be used once, according to Observation 4.2. Since for every $j \in [k_1]$, $1 \leq size_\Delta(t_{s_j}) < n_M$, $1 \leq size_\Delta(t_{i_j}) < n_M$, and $1 \leq size_\Delta(\hat{t}_{s_j}) < n_M$, we know for every $l \in [k_1]$:

$$1 \leq size_\Delta(t^{(l)}) < (2k_1 + 1) \cdot n_M \leq (2k + 1) \cdot n_M$$

Thus every sequence $t^{(l)}$ can overlap at most two parts of successive symbols D in $\underline{tree}(1)$. The k_1 sequences together can overlap at most $2k_1 \leq 2k$ parts of successive symbols D in $\underline{tree}(1)$. Since there are $2k+1$ parts of successive symbols D in $\underline{tree}(1)$, there must exist one subsequence

$$b = BD^{n_M \cdot (2k+1)}B \quad \text{or} \quad b = BD^{n_M \cdot (2k+1)}E$$

of $\underline{tree}(1)$ which completely is a part of $\hat{t}$ or of a tree $\hat{t}_{i_l}$ for some $l \in [k_1]$.

We present an example situation with $k = k_1 = 2$ and with a subsequence $b = B\,D \ldots D\,B$ in $\hat{t}_{i_1}$:

$$\underbrace{B\,D\ldots}_{\hat{t}}\underbrace{\ldots D\,B\,D\ldots}_{t^{(1)}}\underbrace{\ldots D\,\overbrace{B\,D\ldots\ldots D\,B}^{b}\,D\ldots}_{\hat{t}_{i_1}}\underbrace{\ldots D\,B\,D\ldots}_{t^{(2)}}\underbrace{\ldots D\,E}_{\hat{t}_{i_2}}$$

This subsequence b must appear in $\underline{tree}(0)$, because $\underline{tree}(0)$ is built up, using each of the patterns $\hat{t}, \hat{t}_{i_1}, \ldots, \hat{t}_{i_{k_1}}, \hat{t}_{s_1}, \ldots, \hat{t}_{s_{k_1}}$ exactly once by Observation 4.2.

(It is not important for this proof that the relative positions of these patterns can change from $\underline{tree}(1)$ to $\underline{tree}(0)$.)

The patterns $t_{s_1}, \ldots, t_{s_{k_1}}, t_{i_1}, \ldots, t_{i_{k_1}}$ do not appear in $\underline{tree}(0)$ any more. These patterns can only contain symbols D and B, because $size_\Delta(\hat{t}_{i_{k_1}}) \geq 1$ and thus the last symbol E must be a part of $\hat{t}_{i_{k_1}}$.

If there is a symbol B in one of these patterns, then the number of symbols B decreases and thus $\underline{tree}(0) \notin \{(BD^n)^{2k+1}E \mid n \geq 0\}$, contradicting the assumption.

If these patterns only contain symbols D, then the number of symbols D decreases and the number of symbols B is constant. Thus we must have a block $b' = B\ D \ldots D\ B$ or $b' = B\ D \ldots D\ E$ with less than $n_M \cdot (2k+1)$ successive symbols D. Since b and b' have a different number of successive symbols D, we have $\underline{tree}(0) \notin \{(BD^n)^{2k+1}E \mid n \geq 0\}$, contradicting the assumption.

Note that the last steps of the above argumentation need the requirement of M to be producing.

Case 2: $k_1 = k_2 + 1$

In this case we can represent t as follows, where for every $l \in [k_1]$, $t^{(l)}$ is a sequence of patterns taken from the $3k_1 - 1$ patterns $t_{s_1}, \ldots, t_{s_{k_1}}, t_{i_1}, \ldots, t_{i_{k_1-1}}, \hat{t}_{s_1}, \ldots, \hat{t}_{s_{k_1}}$:

$$t = \underline{tree}(1) = \hat{t}\, t^{(1)}\, \hat{t}_{i_1}\, t^{(2)}\, \hat{t}_{i_2} \ldots t^{(k_1-1)}\, \hat{t}_{i_{k_1-1}}\, t^{(k_1)}$$

For every $l \in [k_1 - 1]$, the tree $t^{(l)}$ is built up from at least one pattern, and $t^{(k_1)}$ is built up from at least two patterns. For every $l \in [k_1 - 1]$, the tree $t^{(l)}$ is constructed from at most $2k_1 - 1$ patterns, if the other trees $t^{(l')}$ with $l' \in [k_1 - 1]$ are built up from exactly one pattern and $t^{(k_1)}$ is built up from exactly two patterns, because each pattern can only be used once, according to Observation 4.2. The tree $t^{(k_1)}$ is constructed from at most $2k_1$ patterns, if the other trees $t^{(l')}$ with $l' \in [k_1 - 1]$ are built up from exactly one pattern. Then we can apply the same argumentation as in Case 1. □

Lemma 4.5 For every $k \geq 1$, $\{(BD^n)^{2k}E \mid n \geq 0\} \notin S_{(k)}I_{(k-1)}T_{out}$.

Proof. The proof of this lemma is analogous to the proof of Lemma 4.4. □

The following lemma completes the requirements for the desired hierarchy of attributed tree transducers.

Lemma 4.6

- For every $k \geq 1$, $\{(BD^n)^{2k}E \mid n \geq 0\} \in S_{(k)}I_{(k)}T_{out}$.
- For every $k \geq 0$, $\{(BD^n)^{2k+1}E \mid n \geq 0\} \in S_{(k+1)}I_{(k)}T_{out}$.

Proof. For every $k \geq 1$ we define an *si*-tree transducer $M^{(2k)} = (\mathcal{A}^{(2k)}, \Delta, \Sigma, s_1, root, R^{(2k)})$ with:
$\Delta = \{B^{(1)}, D^{(1)}, E^{(0)}\}$,
$\Sigma = \{\gamma^{(1)}, \alpha^{(0)}\}$,
$\mathcal{A}^{(2k)} = (A^{(2k)}, A_s^{(2k)}, A_i^{(2k)})$ with $A^{(2k)} = \{s_1, \ldots, s_k, i_1, \ldots, i_k\}$,

$A_s^{(2k)} = \{s_1, \ldots, s_k\}$, and $A_i^{(2k)} = \{i_1, \ldots, i_k\}$, and
$R^{(2k)} = R_{root}^{(2k)} \cup R_\gamma^{(2k)} \cup R_\alpha^{(2k)}$ with:

$$\begin{array}{lcll} R_{root}^{(2k)} & = & \{s_1(z) \rightarrow B\, s_1(z1) & \} \cup \\ & & \{i_j(z1) \rightarrow B\, s_{j+1}(z1) & \mid j \in [k-1]\} \cup \\ & & \{i_k(z1) \rightarrow E & \} \\ R_\gamma^{(2k)} & = & \{s_j(z) \rightarrow D\, s_j(z1) & \mid j \in [k]\} \cup \\ & & \{i_j(z1) \rightarrow D\, i_j(z) & \mid j \in [k]\} \\ R_\alpha^{(2k)} & = & \{s_j(z) \rightarrow B\, i_j(z) & \mid j \in [k]\} \end{array}$$

For every $k \geq 0$ we define an *si*–tree transducer
$M^{(2k+1)} = (\mathcal{A}^{(2k+1)}, \Delta, \Sigma, s_1, root, R^{(2k+1)})$ with:
$\Delta = \{B^{(1)}, D^{(1)}, E^{(0)}\}$,
$\Sigma = \{\gamma^{(1)}, \alpha^{(0)}\}$,
$\mathcal{A}^{(2k+1)} = (A^{(2k+1)}, A_s^{(2k+1)}, A_i^{(2k+1)})$ with $A^{(2k+1)} = \{s_1, \ldots, s_{k+1}, i_1, \ldots, i_k\}$,
$A_s^{(2k+1)} = \{s_1, \ldots, s_{k+1}\}$, and $A_i^{(2k+1)} = \{i_1, \ldots, i_k\}$, and
$R^{(2k+1)} = R_{root}^{(2k+1)} \cup R_\gamma^{(2k+1)} \cup R_\alpha^{(2k+1)}$ with:

$$\begin{array}{lcll} R_{root}^{(2k+1)} & = & \{s_1(z) \rightarrow B\, s_1(z1) & \} \cup \\ & & \{i_j(z1) \rightarrow B\, s_{j+1}(z1) & \mid j \in [k]\} \\ R_\gamma^{(2k+1)} & = & \{s_j(z) \rightarrow D\, s_j(z1) & \mid j \in [k+1]\} \cup \\ & & \{i_j(z1) \rightarrow D\, i_j(z) & \mid j \in [k]\} \\ R_\alpha^{(2k+1)} & = & \{s_j(z) \rightarrow B\, i_j(z) & \mid j \in [k]\} \cup \\ & & \{s_{k+1}(z) \rightarrow E & \} \end{array}$$

Clearly, for every $k \geq 1$, $L_{out}(M^{(k)}) = \{(BD^n)^k E \mid n \geq 0\}$. Thus we can conclude the statement of the lemma. □

From Lemma 4.4, Lemma 4.5, and Lemma 4.6 we gain the following hierarchy for classes of output languages of *si*–tree transducers with bounded number of attributes:

Theorem 4.7 $S_{(k)}I_{(k-1)}T_{out} \subset S_{(k)}I_{(k)}T_{out} \subset S_{(k+1)}I_{(k)}T_{out}$, for every $k \geq 1$. □

This theorem can be transformed into the following theorem that presents a hierarchy for classes of tree transformations of *si*–tree transducers with bounded number of attributes (cf. also Figure 11):

Theorem 4.8 $S_{(k)}I_{(k-1)}T \subset S_{(k)}I_{(k)}T \subset S_{(k+1)}I_{(k)}T$, for every $k \geq 1$. □

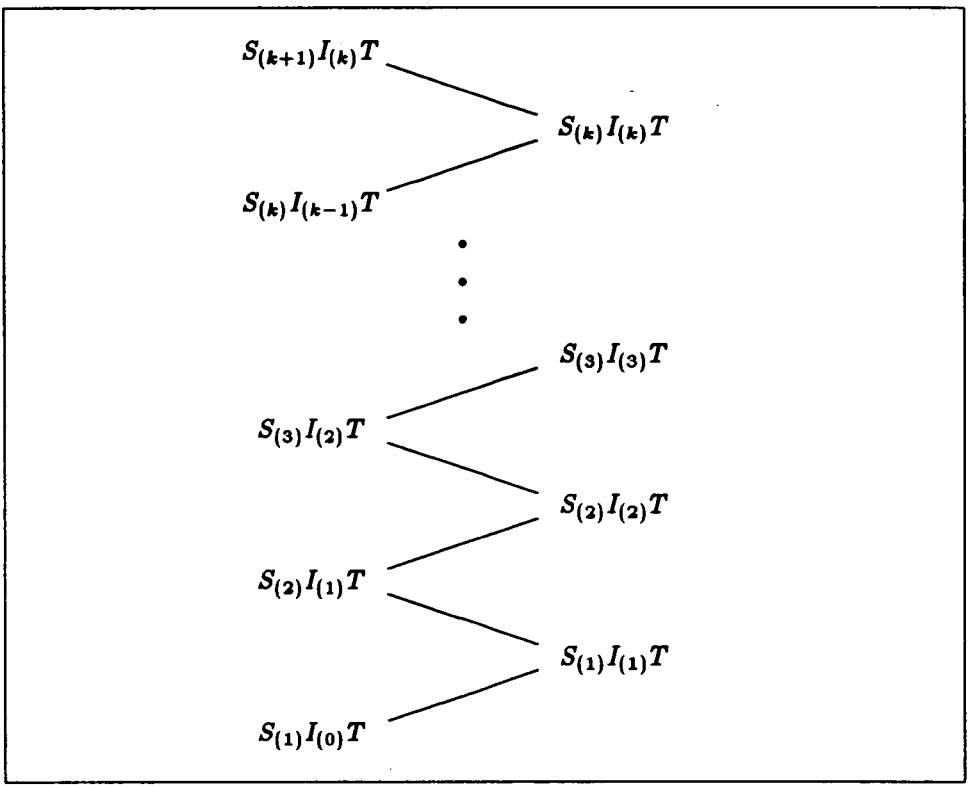


Figure 11: Hierarchy of tree transformation classes.

5 Summary and further research topics

In this paper we have developed a pumping lemma for output languages of noncircular, producing, and visiting attributed tree transducers. We have restricted the applications of the pumping lemma to monadic output languages yielding two results for attributed tree transducers. In particular,

- we have proved that the language $\{B^{2^n}E \mid n \geq 0\}$ can be no output language of a noncircular, producing, and visiting attributed tree transducer, using our pumping lemma together with arithmetic properties of this language, and
- we have proved a hierarchy for noncircular, producing, and visiting attributed tree transducers with bounded number of attributes, using our pumping lemma together with structural properties of languages.

There are several further research topics in the area of pumping lemmata for attributed tree transducers and other kinds of tree transducers:

- Are there non-monadic languages which can be proved not to be output languages of attributed tree transducers with the help of our pumping lemma in

a justifiable expense? In the case of non-monadic languages the proofs become very much harder, because the output patterns can no more be treated like concatenated strings as in the proof of Lemma 4.4. The output patterns are non-monadic trees which occur in a non- monadic output tree. The main problem is to find a complete case analysis for all possibilities to construct an output tree with output patterns. Then we have to derive a contradiction for every case. Additionally we have the difficulty that output patterns can occur more than once in an output tree $\underline{tree}(1)$, as can be seen in Figure 9. Thus in the case of non- monadic output languages there is no helping observation as Observation 4.2.

- A similar pumping lemma as for attributed tree transducers can be developed for macro tree transducers (cf. [EV85]). It will be introduced in another paper which is in preparation (cf. [Küh94]). Is it possible to use this pumping lemma in a proof that the difference set SI_fT–S_fT of subclasses of macro attributed tree transducers is not empty, as it was conjectured in [KV94]?
- As next step it should be possible to construct a pumping lemma for macro attributed tree transducers (cf. [KV94]) as combination of the lemmata for attributed tree transducers and macro tree transducers. Then as special case of it we have a pumping lemma for the class SI_fT and perhaps it is possible to prove that the difference set S_fT–SI_fT is not empty, as it also was conjectured in [KV94].

Acknowledgement
We would like to thank Zoltan Fülöp for carefully reading an earlier version of this paper and for making useful suggestions on its contents.

References

[AU71] A.V. Aho and J.D. Ullman. Translations on a context free grammar. *Inform. and Control*, 19:439–475, 1971.

[BM82] C. Bader and A. Moura. A generalization of Ogden's lemma. *J. Assoc. Comput. Mach.*, 29:404–407, 1982.

[Boc76] G. Bochmann. Semantic evaluation from left to right. *Comm. Assoc. Comput. Sci.*, 19:55–62, 1976.

[BPS61] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Z. Phonetik. Sprach. Komm.*, 14:143–172, 1961.

[BS86a] R. Boonyavatana and G. Slutzki. A generalized Ogden's lemma for linear context-free languages. *Bulletin of the EATCS*, 28:20–26, 1986.

[BS86b] R. Boonyavatana and G. Slutzki. Ogden's lemma for nonterminal bounded languages. *RAIRO*, 20:457–471, 1986.

[Ems91] K. Emser–Loock. Integration von attributierten Grammatiken und primitiv–rekursiven Programmschemata. Master Thesis, RWTH Aachen, 1991.

[Eng75] J. Engelfriet. Bottom–up and top–down tree transformations — a comparison. *Math. Syst. Theory*, 9:198–231, 1975.

[EPR81] A. Ehrenfeucht, R. Parikh, and G. Rozenberg. Pumping lemmas for regular sets. *SIAM J. Comput.*, 10:536–541, 1981.

[ERS80] J. Engelfriet, G. Rozenberg, and G. Slutzki. Tree transducers, L systems, and two–way machines. *J. Comput. Syst. Sci.*, 20:150–202, 1980.

[Ési80] Z. Ésik. Decidability results concerning tree transducers. *Acta Cybernetica*, 5:1–20, 1980.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31:71–145, 1985.

[Fül81] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[FV91] Z. Fülöp and S. Vágvölgyi. Attributed tree transducers cannot induce all deterministic bottom–up tree transformations. manuscript, to appear in Information and Computation, 1991.

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25:355–423, 1988.

[GS83] F. Gécseg and M. Steinby. *Tree Automata.* Akademiai Kiado, Budapest, 1983.

[Hab89] A. Habel. *Hyperedge replacement: grammars and languages.* PhD thesis, University of Bremen, 1989.

[Hin90] F. Hinz. *Erzeugung von Bildsprachen durch Chomsky–Grammatiken — Entscheidbarkeits– und Komplexitätsfragen.* PhD thesis, RWTH Aachen, 1990.

[Knu68] D.E. Knuth. Semantics of context–free languages. *Math. Syst. Theory*, 2:127–145, 1968.

[Kre79] H.-J. Kreowski. A pumping lemma for context–free graph languages. *Lect. Not. Comp. Sci.*, 73:270–283, 1979.

[Küh94] A. Kühnemann. A pumping lemma for output languages of macro tree transducers. Technical report, Technical University of Dresden, 1994. in preparation.

[Kus91] S. Kuske. Ein Pumping–Lemma für Kantenersetzungssprachen bezüglich maximaler Weglänge. Master Thesis, University of Bremen, 1991.

[Kus93] S. Kuske. A maximum path length pumping lemma for edge–replacement languages. In *FCT'93*, pages 342–351. Springer-Verlag, 1993. LNCS 710.

[KV94] A. Kühnemann and H. Vogler. Synthesized and inherited functions — a new computational model for syntax–directed semantics. *Acta Informatica*, 31:431–477, 1994.

[Ogd68] W. Ogden. A helpful result for proving inherent ambiguity. *Math. Syst. Theory*, 2:191–194, 1968.

[Per76] C.R. Perrault. Intercalation lemmas for tree transducer languages. *J. Comput. Syst. Sci.*, 13:246–277, 1976.

[Rou70] W.C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory*, 4:257–287, 1970.

[Sch60] S. Scheinberg. Note on the boolean properties of context free languages. *Inform. and Control*, 3:372–375, 1960.

[Tha70] J.W. Thatcher. Generalized^2 sequential machine maps. *J. Comput. Syst. Sci.*, 4:339–367, 1970.

[Wis76] D. S. Wise. A strong pumping lemma for context–free languages. *Theoret. Comp. Sci.*, 3:359–369, 1976.

[Yu89] S. Yu. A pumping lemma for deterministic context–free languages. *Inform. Proc. Letters*, 31:47–51, 1989.

Received March 26, 1994

Acta Cybernetica, Vol. 11, No. 4, Szeged, 1994

On Semi-Conditional Grammars with Productions Having either Forbidding or Permitting Conditions

A. Meduna * A. Gopalaratnam*

Abstract

This paper simplifies semi-conditional grammars so their productions have no more than one associated word–either a permitting condition or a forbidding condition. It is demonstrated that this simplification does not decrease the power of semi-conditional grammars.

1 Introduction

A semi-conditional grammar is a context-free grammar with productions having two associated words–a permitting condition and a forbidding condition. Such a production can rewrite a word, w, provided its permitting/forbidding condition is/is not a subword of w. Semi-conditional grammars without erasing productions characterize the family of context-sensitive languages; when erasing productions are allowed, these grammars define all family of recursively enumerable languages.

This paper studies a simplified concept of these grammars, whose productions have no more than one associated word–either a permitting condition or a forbidding condition. It is shown that this simplification does not decrease the generative power of semi-conditional grammars.

2 Definitions and Examples

We assume that the reader is familiar with formal language theory (see [3]).

Let V be an alphabet V^* denotes the free monoid generated by V under the operation of concatenation, where λ denotes the unit of V^*. Let $V^+ = V^* - \{\lambda\}$. Given a word, $w \in V^*$, $|w|$ represents the length of w, and $alph(w)$ denotes the set of symbols occurring in w. We set $sub(w) = \{y : y \text{ is a subword of } w\}$. Given a symbol, $a \in V, \#_a w$ denotes the number of occurrences of a in w.

A semi-conditional grammar (an *sc*-grammar for short) is a quadruple, $G = (V, P, S, T)$, where V, T, and S are the total alphabet, the terminal alphabet ($T \subset V$), and the axiom, respectively, and P is a finite set of productions of the form $(A \to \alpha, \beta, \mu)$ with $A \in V - T, \alpha \in V^*, \beta \in V^+ \cup \{0\}$, and $\mu \in V^+ \cup \{0\}$, where 0

*Department of Computer Science, University of Missouri-Columbia, Columbia, Missouri 65211, USA

is a special symbol, $0 \notin V$ (intuitively, 0 means that the production's condition is missing). If $(A \to \alpha, \beta, \mu) \in P$ implies $\alpha \neq \Lambda$, G is said to be propagating. G has degree $(i, 0)$, where i is a natural number, if for every $(A \to \alpha, \beta, \mu) \in P, \beta \in V^+$ implies $|\beta| \leq i$, and $\mu = 0$. G has degree $(0, j)$, where j is a natural number, if for every $(A \to \alpha, \beta, \mu) \in P, \beta = 0$, and $\mu \in V^+$ implies $|\mu| \leq j$. G has degree (i, j), where i and j are two natural numbers, if for every $(A \to \alpha, \beta, \mu) \in P, \beta \in V^+$ implies $|\beta| \leq i$, and $\mu \in V^+$ implies $|\mu| \leq j$. Let $u, v \in V^*$, and $(A \to \alpha, \beta, \mu) \in P$. Then, u directly derives v according to $(A \to \alpha, \beta\mu)$, denoted by

$$u \Rightarrow v \; [(A \to \alpha, \beta, \mu)]$$

provided for some $u_1, u_2 \in V^*$, the following conditions (1) through (4) hold

(1) $u = u_1 A u_2$
(2) $v = u_1 \alpha u_2$
(3) $\beta \neq 0$ implies $\beta \in \text{sub}(u)$
(4) $\mu \neq 0$ implies $\mu \notin \text{sub}(u)$

When no confusion exists, we simply write $u \Rightarrow v$. As usual, we extend $\Rightarrow$ to $\Rightarrow^i$ (where $i \geq 0$), $\Rightarrow^+$, and $\Rightarrow^*$. The language of G, denoted by $L(G)$, is defined by $L(G) = \{w \in T^*; S \Rightarrow^* w\}$.

Now, we introduce the central notion of this paper–a simple semi-conditional grammar. Informally, a simple semi-conditional grammar is an *sc*-grammar in which any production has no more than one condition–either a permitting condition or a forbidding condition. Formally, let $G = (V, P, S, T)$ be an *sc*-grammar. G is a *simple semi-conditional grammar* (an ssc-grammar for short) if $(A \to x, \alpha, \beta) \in P$ implies $\{0\} \subseteq \{\alpha, \beta\}$.

To give an insight into *ssc* grammars, let us present two examples.

Example 1 Let

$$G = (\{S, A, X, C, Y, a, b\}, P, S, \{a, b\})$$

be an *ssc*-grammar, where

$$\begin{aligned} P \;=\; & \{(S \to AC, 0, 0), \\ & (A \to aXb, Y, 0), \\ & (C \to Y, A, 0), \\ & (Y \to Cc, 0, A), \\ & (A \to ab, Y, 0), \\ & (Y \to c, 0, A), \\ & (X \to A, C, 0)\} \end{aligned}$$

Notice that G is propagating, and it has degree (1, 1). Consider $aabbcc$. G derives this word as follows:

$$S \Rightarrow AC \Rightarrow AY \Rightarrow aXbY \Rightarrow aXbCc \Rightarrow aAbCc \Rightarrow aAbYc \Rightarrow aabbYc \Rightarrow aabbcc.$$

Obviously,

$$L(G) = \{a^n b^n c^n; n \geq 1\}.$$

Note that $\{a^n b^n c^n; n \geq 1\}$ is not a context-free language.

Example 2 Let

$$G = (\{S, A, B, X, Y, a\}, P, S, \{a\})$$

be an *ssc*-grammar, where P is defined as follows:

$$\begin{aligned} P = \{&(S \to a, 0, 0),\\ &(S \to X, 0, 0),\\ &(X \to YB, 0, A),\\ &(X \to aB, 0, A),\\ &(Y \to XA, 0, B),\\ &(Y \to aA, 0, B),\\ &(A \to BB, XA, 0)\}\\ &(B \to AA, YB, 0)\}\\ &(B \to a, a, 0)\}. \end{aligned}$$

G is a propagating *ssc*-grammar of degree (2,1). For *aaaaaaaa*, G makes the following derivation:

$$\begin{aligned} S \Rightarrow\ & X \Rightarrow YB \Rightarrow YAA \Rightarrow XAAA \Rightarrow XABBA \Rightarrow XABBBB \Rightarrow XBBBBBB \Rightarrow\\ & aBBBBBB \Rightarrow aBaBBBBB \Rightarrow aBaBBBBa \Rightarrow aaaBBBBa \Rightarrow\\ & aaaBBBaa \Rightarrow aaaaBaaa \Rightarrow aaaaaaaa. \end{aligned}$$

Clearly, G generates $\{a^{2^n}; n \geq 0\}$, that is,

$$L(G) = \{a^{2^n}; n \geq 0\}.$$

Note that $\{a^{2^n}; n \geq 0\}$ is not context-free.

The family of languages generated by *ssc*-grammars of degree (i, j) is denoted by $\mathbf{SSC}(i, j)$. Set

$$\mathbf{SSC} = \bigcup_{i=0}^{\infty} \bigcup_{j=0}^{\infty} \mathbf{SSC}(i, j).$$

To indicate that only propagating grammars are considered, we use the prefix **prop-**; for intance, **prop-SSC** (2, 1) denotes the family of languages generated by propagating *ssc*-grammars of degree (2, 1).

The families of context-free, context-sensitive, and recursively enumerable languages are denoted by **CF**, **CS**, and **RE**, respectively.

Let us finally recall that a context sensitive grammar in Penttonen normal form is a quadruple, $G = (V, P, S, T)$, where V, S, and T have the same meaning as for an *sc*-grammar, and any production in P is either of the form $AB \to AC$ or of the form $A \to \alpha$, where $A, B, C \in V - T, \alpha \in (T \cup (V - T)^2)$ (see [2]). In the standard manner, we define $\Rightarrow, \Rightarrow^i, \Rightarrow^+, \Rightarrow^*$, and $L(G)$. If we want to express that $x \Rightarrow y$ in G according to $p \in P$, we write $x \Rightarrow y\ [p]$.

3 Results

From the definition, the results achieved in [1], and the examples given in the previous section, we see that

$$\mathbf{CF} \subset \mathbf{prop\text{-}SSC} \subseteq \mathbf{prop\text{-}SC} = \mathbf{prop\text{-}SC}(2,1) = \mathbf{prop\text{-}SC}(1,2) = \mathbf{CS}$$

and

$$\mathbf{prop\text{–}SSC} \subseteq \mathbf{SSC} \subseteq \mathbf{SC} = \mathbf{SC}(2,1) = \mathbf{SC}(1,2) = \mathbf{RE}$$

This section states that

$$\mathbf{CF}$$

$$\subset$$

$$\mathbf{prop-SSC} = \mathbf{prop-SSC}(2,1) = \mathbf{prop-SSC}(1,2) =$$

$$\mathbf{prop-SC} = \mathbf{prop-SC}(2,1) = \mathbf{prop-SC}(1,2) = \mathbf{CS}$$

$$\subset$$

$$\mathbf{SSC} = \mathbf{SSC}(2,1) = \mathbf{SSC}(1,2) = \mathbf{SC} = \mathbf{SC}(2,1) = \mathbf{SC}(1,2) = \mathbf{RE}$$

In other words, we demonstrate that *ssc*-grammars are as powerful as *sc*-grammars. To establish this result, we first prove that propagating *ssc*-grammars of degree (2,1) generate precisely the family of context-sensitive languages.

Theorem 1 $\mathbf{CS} = \mathbf{prop-SSC}(2,1)$.

Proof. Clearly, $\mathbf{prop-SSC}(2,1) \subseteq \mathbf{CS}$, so it suffices to prove the converse inclusion.

Let $G = (V, P, S, T)$ be a context-sensitive grammar in Penttonen normal form. We construct an *ssc*-grammar, $G' = (V \cup W, P', S, T)$, that generates $L(G)$. Let

$$W = \{\tilde{B}; AB \to AC \in P, A, B, C \in V - T\}$$

We define P' in the following way:

1. if $A \to \alpha \in P, A \in V - T, \alpha \in T \cup (V - T)^2$,
   then add $(A \to \alpha, 0, 0)$ into P',
2. if $AB \to AC \in P, A, B, C \in V - T$,
   then add

$$(B \to \tilde{B}, 0, \tilde{B}), (\tilde{B} \to C, A\tilde{B}, 0), \text{ and } (\tilde{B} \to B, 0, 0)$$

   to P'($\tilde{B}$ is the $\sim$ version of B in $AB \to AC$).

Notice that G is a propagating *ssc*-grammar of degree (2,1). Moreover, from (2), we have for any $\tilde{B} \in W$

$$S \Rightarrow_{G'}^{*} \alpha \text{ implies } \#_{\tilde{B}}\alpha \leq 1$$

because the only production that can generate $\tilde{B}$ is of the form $(B \to \tilde{B}, \emptyset, \tilde{B})$.

Let g be the finite substitution from V^* into $(W \cup V)^*$ defined as follows: for all $D \in V$,

1. if $\tilde{D} \in W$($\tilde{D}$ is the $\sim$ version of D), then $g(D) = \{D, \tilde{D}\}$;
2. if $\tilde{D} \notin W$, then $g(D) = \{D\}$.

Next, we will show that for any $w \in V^+$,

$$S \Rightarrow_G^m w \text{ if and only if } S \Rightarrow_{G'}^n v \text{ with } v \in g(w)$$

for some $m, n \geq 0$.

Only if: This is proved by induction on m.

Basis: Let $m = 0$. The only w is S as $S \Rightarrow_G^0 S$. Clearly, $S \Rightarrow_{G'}^n S$ for $n = 0$, and $S \in g(S)$.

Induction Hypothesis: Assume that the claim holds for all derivations of length m or less, for some $m \geq 0$.

Induction Step: Consider a derivation $S \Rightarrow_G^{m+1} \alpha, \alpha \in V^+$. Because $m+1 \geq 1$, there is some $\beta \in V^*$ and $p \in P$ such that $S \Rightarrow_G^m \beta \Rightarrow_G \alpha\ [p]$. By the induction hypothesis, $S \Rightarrow_{G'}^n \beta'$ for some $\beta' \in g(\beta)$ and $n \geq 0$. Next, we distinguish two cases, case (i) considers p with one nonterminal on its left-hand side, and case (ii) considers p with two nonterminals on its left-hand side.

(i) Let $p = D \rightarrow \beta_2 \in P, D \in V - T, \beta_2 \in T \cup (V-T)^2, \beta = \beta_1 D \beta_3, \beta_1, \beta_3 \in V^*$, $\alpha = \beta_1\beta_2\beta_3, \beta' = \beta_1' X \beta_3', \beta_1' \in g(\beta_1), \beta_3' \in g(\beta_3)$, and $X \in g(D)$. By (1), $(D \rightarrow \beta_2, 0, 0) \in P$. If $X = D$, then $S \Rightarrow_{G'}^n \beta_1' D \beta_3' \Rightarrow_{G'} \beta_1'\beta_2\beta_3'\ [(D \rightarrow \beta_2, 0, 0)]$. Because $\beta_1' \in g(\beta_1), \beta_3' \in g(\beta_3)$, and $\beta_2 \in g(\beta_2)$, we obtain $\beta_1'\beta_2\beta_3' \in g(\beta_1\beta_2\beta_3) = g(\alpha)$. If $X = \tilde{D}$, we have $(X \rightarrow D, 0, 0) \in P'$, so $S \Rightarrow_{G'}^n \beta_1' X \beta_3' \Rightarrow_{G'} \beta_1' D \beta_3'\ [(D \rightarrow \beta_2, 0, 0)]$, and $\beta_1'\beta_2\beta_3' \in g(\alpha)$.

(ii) Let $p = AB \rightarrow AC \in P, A, B, C \in V - T, \beta = \beta_1 AB \beta_2, \beta_1, \beta_2 \in V^*$, $\alpha = \beta_1 AC \beta_2, \beta' = \beta_1' XY \beta_2', \beta_1' \in g(\beta_1), \beta_2' \in g(\beta - 2), X \in g(A)$, and $Y \in g(B)$. Recall that for any $\tilde{B}$, $\#_{\tilde{B}} \beta' \leq 1$ and $(\tilde{B} \rightarrow B, 0, 0) \in P'$. Then, $\beta' \Rightarrow_{G'}^i \bar{\beta}_1 AB \bar{\beta}_2$ for some $i \in \{0, 1\}$ so $\bar{\beta}_j \in g(\beta_j), j = 1, 2$, and $(g(A) \cup g(B)) \cap alph(\beta_1 AB\beta_2) = \{A, B\}$. At this point, we have:

$$\begin{aligned} S &\Rightarrow_{G'}^* \bar{\beta}_1 AB \bar{\beta}_2 \\ &\Rightarrow_{G'} \bar{\beta}_1 A \tilde{B} \bar{\beta}_2 \quad [(B \rightarrow \tilde{B}, 0, \tilde{B})] \\ &\Rightarrow_{G'} \bar{\beta}_1 AC \bar{\beta}_2 \quad [(\tilde{B} \rightarrow C, A\tilde{B}, 0)] \end{aligned}$$

where $\bar{\beta}_1 \in g(\beta_1), \bar{\beta}_2 \in g(\beta_2), C \in g(C)$, i.e., $\bar{\beta}_1 AC \bar{\beta}_2 \in g(\alpha)$.

If: This is established by induction on n; in other words, we demonstrate that

$$\text{if } S \Rightarrow_{G'}^n v \text{ with } v \in g(w) \text{ for some } w \in V^+, \text{ then } S \Rightarrow_G^* w.$$

Basis: For $n = 0, v$ surely equals S as $S \Rightarrow_{G'}^0 S$. Because $S \in g(S)$, we have $w = S$. Clearly, $S \Rightarrow_G^0 S$.

Induction Hypothesis: Assume the claim holds for all derivations of length n or less, for some $n \geq 0$.

Induction Step: Consider a derivation, $S \Rightarrow_G^{n+1} \alpha', \alpha' \in g(\alpha), \alpha \in V^+$. As $n+1 \geq 1$, there exists some $\beta \in V^+$ such that $S \Rightarrow_{G'}^n \beta' \Rightarrow_{G'} \alpha'\ [p], \beta' \in g(\beta)$. By induction hypothesis, $S \Rightarrow_G^* \beta$. Let $\beta' = \beta_1' B' \beta_2', \beta = \beta_1 B \beta_2, \beta_j' \in g(\beta_j), j = 1, 2, \beta_j \in V^*, B' \in g(B), B \in V - T, \alpha' = \beta_1' \mu' \beta_2'$, and $p = (B' \rightarrow \mu', \mu_1, \mu_2) \in P'$. The following three cases — (i), (ii), and (iii) — cover all possible forms of the derivation step $\beta' \Rightarrow_{G'} \alpha'\ [p]$.

(i) $\mu' \in g(B)$. Then, $S \Rightarrow_G^* \beta_1 B \beta_2, \beta_1' \mu' \beta_2' \in g(\beta_1 B \beta_2)$, i.e., $\alpha' \in g(\beta_1 B \beta_2)$.

(ii) $B' = B \in V - T, \mu' \in T \cup (V-T)^2, \mu_1 = 0 = \mu_2$. Then, there exists a production, $B \to \mu' \in P$, so $S \Rightarrow_G^* \beta_1 B \beta_2 \Rightarrow_G \beta_1 \mu' \beta_2 [B \to \mu']$. Since $\mu' \in g(\mu')$, we have $\alpha = \beta_1 \mu' \beta_2$ such that $\alpha' \in g(\alpha)$.

(iii) $B' = \tilde{B}, \mu' = C, \mu_1 = A\tilde{B}, \mu_2 = 0, A, B, C \in V - T$. Then, there exists a production of the form $AB \to AC \in P$. Since $\#_Z \beta' \leq 1, Z = \tilde{B}$, and $A\tilde{B} \in sub(\beta')$, we have $\beta_1' = \delta' A, \beta_1 = \delta A$ (for some $\delta \in V^*$), and $\delta' \in g(\delta)$. Thus, $S \Rightarrow_G \delta AB\beta_2 \Rightarrow_G \delta AC\beta_2 [AB \to AC], \delta AC\beta_2 = \beta_1 C \beta_2$. Because $C \in g(C)$, we get $\alpha = \beta_1 C \beta_2$ such that $\alpha' \in g(\alpha)$.

By the principle of induction, we have thus established that for any $w \in V^+, S \Rightarrow_G^* w$ if and only f $S \Rightarrow_{G'}^* v$ with $v \in g(w)$. Because $g(x) = \{x\}$, for any $x \in T^*$, we have for every $w \in T^+$,

$$S \Rightarrow_G^* w \text{ if and only if } S \Rightarrow_{G'}^* w.$$

Thus, $L(G) = L(G')$, and the theorem holds. Q.E.D.

Corollary 2 $\mathbf{CS} = \mathbf{prop-SSC}(2,1) = \mathbf{prop-SSC} = \mathbf{prop-SC}(2,1) = \mathbf{prop-SC}$.

We now turn to the investigation of *ssc*-grammars with erasing productions. We prove that these grammars generate precisely the family of recursively enumerable languages.

Theorem 3 $\mathbf{RE} = \mathbf{SSC}$ *(2,1)*.

Proof. Clearly, we have the containment $\mathbf{SSC}(2,1) \subseteq \mathbf{RE}$; hence, it suffices to show $\mathbf{RE} \subseteq \mathbf{SSC}(2,1)$. Every language $L \in \mathbf{RE}$ can be generated by a recursively enumerable grammar, whose productions are of the form $AB \to AC$ or $A \to \alpha$ where $A, B, C \in V - T, \alpha \in T \cup (V-T)^2 \cup \{\lambda\}$ (see [2]). Thus, the containment $\mathbf{RE} \subseteq \mathbf{SSC}(2,1)$ can be proved by analogy with the proof of Theorem 1 (the details are left to the reader). Q.E.D.

Corollary 4 $\mathbf{RE} = \mathbf{SSC}(2,1) = \mathbf{SSC} = \mathbf{SC}(2,1) = \mathbf{SC}$.

To demonstrate that propagating *ssc*-grammars of degree (1,2) characterize CS, we first establish a normal form for context-sensitive grammars (see Lemmas 5 and 6).

Lemma 5 Every $L \in CS$ can be generated by a context sensitive grammar, $G = (N_{CF} \cup N_{CS} \cup T, P, S, T)$, where N_{CF}, N_{CS}, and T are pairwise disjoint alphabets, and every production in P is either of the form $AB \to AC$ or $A \to x$, where $B \in N_{CS}, A, C \in N_{CF}, x \in N_{CS} \cup T \cup (\cup_{i=1}^2 N_{CF}^i)$.

Proof. Let $L \in \mathbf{CS}$. Without loss of generality, we can assume that L is generated by a context sensitive grammar $G' = (V, P', S, T)$ in Penttonen normal form, that is, every production in P' is either of the form $AB \to AC$ or $A \to BC$ or $A \to a$ (where $A, B, C \in V' - T$ and $a \in T$).

Let $G = (N_{CF} \cup N_{CS} \cup T, P, S, T)$ be the context sensitive grammar defined as follows:

$$\begin{aligned} N_{CF} &= V - T; \\ N_{CS} &= \{\tilde{B}; \tilde{B} \text{ is the tilde version of } B \text{ in } AB \to AC \in P'\}; \\ P &= \{A \to x;\ A \to x \in P',\ A \in V - T,\ x \in T \cup (V-T)^2\} \\ &\quad \cup \{B \to \tilde{B}, \tilde{B} \to AC; AB \to AC \in P', A, B, C \in V - T\}. \end{aligned}$$

Obviously, $L(G') = L(G)$, and G is of the required form. Hence, the lemma holds. Q.E.D.

Lemma 6 Every $L \in CS$ can be generated by a context sensitive grammar $G = (\{S\} \cup N_{CF} \cup N_{CS} \cup T, P, S, T)$, where $\{S\}, N_{CF}, N_{CS}, T$ are pairwise disjoint alphabets, and every production in P is either of the form $S \to aD$ or $AB \to AC$ or $A \to x$, where $a \in T, D \in N_{CF} \cup \{\lambda\}, B \in N_{CS}, A, C \in N_{CF}, x \in N_{CS} \cup T \cup (\cup_{i=1}^{2} N_{CF}^{i})$.

Proof. Let L be a context sensitive language over an alphabet, T. Without loss of generality, we can express L as $L = L_1 \cup L_2$, where $L_1 \subseteq T$ and $L_2 \subseteq TT^+$. Thus, by analogy with the proofs of Theorems 1 and 2 in [2], L_2 can be represented as $L_2 = \cup_{a \in T} aL_a$, where each L_a is a context sensitive language. Let L_a be generated by a context sensitive grammar, $G_a = (N_{CF_a} \cup N_{CS_a} \cup T, P_a, S_a, T)$, of the form of Lemma 5. Clearly, we can assume that for all $a's$, the nonterminal alphabets $(N_{CF_a} \cup N_{CS_a})$ are pairwise disjoint. Let S be a new start symbol. Consider the context sensitive grammar

$$G = (\{S\} \cup N_{CF} \cup N_{CS} \cup T, P, S, T)$$

defined as:

$$N_{CF} = \cup_{a \in T} N_{CF_a};$$
$$N_{CS} = \cup_{a \in T} N_{CS_a};$$
$$P = \cup_{a \in T} P_a \cup \{S \to aS_a; a \in T\} \cup \{S \to a; a \in L_1\}.$$

Obviously, G satisfies the required form, and we have

$$L(G) = L_1 \cup (\cup_{a \in T} aL(G_a)) = L_1 \cup (\cup_{a \in T} aL_a) = L_1 \cup L_2 = L.$$

Consequently, the lemma holds. Q.E.D.

We are now ready to characterize **CS** by propagating *ssc*-grammars of degree (1,2).

Theorem 7 $\mathbf{CS} = \mathbf{prop} - \mathbf{SSC}(1,2)$.

Proof. Clearly, $\mathbf{prop} - \mathbf{SSC}(1,2) \subseteq \mathbf{CS}$; hence, it suffices to prove the converse inclusion.

Let L be a context sensitive language. Without loss of generality, we can assume that L is generated by a context sensitive grammar, $G = (\{S\} \cup N_{CF} \cup N_{CS} \cup T, P, S, T)$, of the form of Lemma 6. Set $V = (\{S\} \cup N_{CF} \cup N_{CS} \cup T)$. Let q be the cardinality of V; $q \geq 1$. Furthermore, let f be an (arbitrary, but fixed) bijection from V onto $\{1, \ldots, q\}$, and let f^{-1} be the inverse of f.

Let $G^{\sim} = (V^{\sim}, P^{\sim}, S, T)$ be a propagating *ssc*-grammar of degree (1,2), in which

$$V^{\sim} = (\cup_{i=1}^{4} W_i) \cup V$$

where

$$W_1 = \{< a, AB \to AC, j >; a \in T, AB \to AC \in P, A, C \in N_{CF}, B \in N_{CS}, 1 \leq j \leq 5\};$$
$$W_2 = \{[a, AB \to AC, j]; a \in T, AB \to AC \in P, A, C, \in N_{CF}, B \in N_{CS}, 1 \leq j \leq q+3\};$$
$$W_3 = \{\hat{B}, B', B''; B \in N_{CS}\};$$
$$W_4 = \{\bar{a}; a \in T\}$$

and $P^{\sim}$ is defined as follows:

1. if $S \to aA \in P, a \in T, A \in (N_{CF} \cup \{\lambda\})$,
   then add $(S \to \bar{a}A, 0, 0)$ to $P^{\sim}$;

2. if $a \in T, A \to x \in P, A \in N_{CF}, x \in (V = \{S\}) \cup (N_{CF})^2$,
   then add $(A \to x, \bar{a}, 0)$ to $P^{\sim}$;

3. if $a \in T, AB \to AC \in P, A, C, \in N_{CF}, B \in N_{CS}$,
   then add to $P^{\sim}$ the following set of productions
   (an informal explanation of these productions can be found below):

$$
\begin{aligned}
&\{(\bar{a} \to < a, AB \to AC, 1 >, 0, 0),\\
&(B \to B', < a, AB \to AC, 1 >, 0),\\
&(B \to \hat{B}, < a, AB \to AC, 1 >, 0),\\
&(< a, AB \to AC, 1 > \to < a, A \to AC, 2 >, 0, B),\\
&(\hat{B} \to B'', 0, B''),\\
&(< a, AB \to AC, 2 > \to < a, AB \to AC, 3 >, 0, \hat{B}),\\
&(B'' \to [a, AB \to AC, 1], < a, AB \to AC, 3 >, 0)\}\\
\cup\ &\{([a, AB \to AC, j] \to [a, AB \to AC, j+1], 0,\\
&f^{-1}(j)[a, AB \to AC, j]); 1 \le j \le q, f(A) \ne j\}\\
\cup\ &\{([a, AB \to AC, f(A)] \to [a, AB \to AC, f(A)+1], 0, 0),\\
&([a, AB \to AC, q+1] \to [a, AB \to AC, q+2], 0,\\
&B'[a, AB \to AC, q+1]),\\
&([a, AB \to AC, q+2] \to [a, AB \to AC, q+3], 0,\\
&< a, AB \to AC, 3 > [a, AB \to AC, q+2]),\\
&(< a, AB \to AC, 3 > \to < a, AB \to AC, 4 >,\\
&[a, AB \to AC, q+3], 0),\\
&(B' \to B, < a, AB \to AC, 4 >, 0),\\
&(< a, AB \to AC, 4 > \to < a, AB \to AC, 5 >, 0, B'),\\
&([a, AB \to AC, q+3] \to C, < a, AB \to AC, 5 >, 0),\\
&(< a, AB \to AC, 5 > \to \bar{a}, 0, [a, AB \to AC, q+3])\}\\
&(B', \hat{B}, \text{ and } B'' \text{ correspond to } B \text{ in } AB \to AC);
\end{aligned}
$$

(4) if $a \in T$, then add $(\bar{a} \to a, 0, 0)$ to $P^{\sim}$.

Let us informally explain the basic idea behind point (3)–the heart of all construction. The production introduced in this point simulate the application of productions of the form $AB \to AC$ in G as follows: an occurrence of B is chosen, and its left neighbor is checked *not* to belong to $V^{\sim} - \{A\}$; at this point, the left neighbor necessarily equals A, so B is rewritten with C.

Formally, we define a finite letter-to-letters substitution g from V^* into $(V^{\sim})^*$ as follows:

if $D \in V$, then add D to $g(D)$;
if $< a, AB \to AC, j > \in W_1 (a \in T, AB \to AC \in P, B \in N_{CS}, A, C \in N_{CF}$,
$j \in \{1, \ldots 5\})$, then add $< a, AB \to AC, j >$ to $g(a)$;
if $[a, AB \to AC, j] \in W_2 (a \in T, AB \to AC \in P, B \in N_{CS}, A, C \in N_{CF}$,
$j \in \{1, \ldots, q+3\})$, then add $[a, AB \to AC, j]$ to $g(B)$;
if $\{\hat{B}, B', B''\} \subseteq W_3 (B \in N_{CS})$, then include $\{\hat{B}, B', B''\}$ to $g(B)$;
if $\bar{a} \in W_4 (a \in T)$, then add $\bar{a}$ to $g(a)$.

Let g^{-1} be the inverse of g.
To show that $L(G) = L(G^{\sim})$, we first prove three claims.

Claim 1: $S \Rightarrow^{+} x$ in $G, x \in V^{*}$, implies $x \in T(V - \{S\})^{*}$.

Proof of Claim 1.
Observe that the start symbol, S, does not appear on the right side of any production and that $S \to x \in P$ implies $x \in T \cup T(V - \{S\})$. Hence, the claim holds.

Claim 2: If $S \Rightarrow^{+} x$ in $G^{\sim}, x \in (V^{\sim})^{*}$, then x has one of the following seven forms:

(i) $x = ay$, where $a \in T, y \in (V - \{S\})^{*}$;

(ii) $x = \bar{a}y$, where $\bar{a} \in W_4, y \in (V - \{S\})^{*}$;

(iii) $x =< a, AB \to AC, 1 > y$, where $< a, AB - AC, 1 > \in W_1$,
$y \in ((V - \{S\}) \cup \{B', \hat{B}, B''\})^{*}, \#_{B''} y \leq 1$;

(iv) $x =< a, AB \to AC, 2 > y$, where $<, AB \to AC, 2 > \in W_1$,
$y \in ((V - \{S, B\}) \cup \{B', \hat{B}, B'\})^{*}, \#_{B'} \leq 1$;

(v) $x =< a, AB \to AC, 3 > y$, where $< a, AB \to AC, 3 > \in W_1$,
$y \in ((V - \{S, B\}) \cup \{B'\})^{*} (\{[a, AB \to AC, j]; 1 \leq j \leq q + 3\} \cup$
$\{\lambda, B''\})((V - \{S, B\}) \cup \{B'\})^{*}$;

(vi) $x =< a, AB \to AC, 4 > z$, where $< a, AB \to AC, 4 > \in W_1$,
$y \in ((V - \{S\}) \cup \{B'\})^{*}[a, AB \to AC, q + 3]((V - \{S\}) \cup \{B'\})^{*}$;

(vii) $x =< a, AB \to AC, 5 > y$ where $< a, AB \to AC, 5 > \in W_1$,
$y \in (V - \{S\})^{*}\{[a, AB \to AC, q_3], \lambda\}(V - \{S\})^{*}$.

Proof of Claim 2.
The claim is proved by induction on the length of derivations.

Basis: Consider $S \Rightarrow x$. By inspection of the productions, we have $S \Rightarrow \bar{a}A\,[(S \to \bar{a}A, 0, 0)]$ for some $\bar{a} \in W_4, A \in (\{\lambda\} \cup N_{CF})$. Therefore, $x = \bar{a}$ or $x = \bar{a}A$ (where $\bar{a} \in W_4$ and $A \in (\{\lambda\} \cup N_{CF})$); in either case, x is a word of the required form.

Induction hypothesis: Assume the claim holds for all derivations of length at most n, for some $n \geq 1$.

Induction step: Consider a derivation of the form $S \Rightarrow^{n+1} x$. Since $n \geq 1$, we have $n + 1 \geq 2$. Thus, there is some z of the required form ($z \in (V^{\sim})^{*}$) such that $S \Rightarrow^{n} z \Rightarrow x\,[p]$ for some $p \in P^{\sim}$.

Let us first prove by contradiction that the first symbol of z does not belong to T. Assume that the first symbol of z belongs to T. As z is of the required form, we have $z = ay$ for some $a \in (V - \{S\})^{*}$. By inspection of $P^{\sim}$, there is no $p \in P^{\sim}$ such that $ay \Rightarrow x[p]$, where $x \in (V^{\sim})^{*}$. We have thus obtained a contradiction, so the first symbol of z is not in T.

Because the first symbol of z does not belong to T, z cannot have form (i); as a result, z has one of forms (ii) through (vii). The following cases I through VI demonstrate that if z has one of these six forms, then x (in $S \Rightarrow^{n} z \Rightarrow x[p]$) has one of the required forms, too.

I. Assume that z is of form (ii), i.e., $z = \bar{a}y, \bar{a} \in W_4$, and $y \in (V - \{S\})^*$. By inspection of the productions in $P^\sim$, we see that p has one of the following forms (a), (b), and (c):

(a) $p = (A \to u, \bar{a}, 0)$ where $A \in N_{CF}$ and $u \in (V - \{S\}) \cup (N_{CF})^2$;
(b) $p = (\bar{a} \to < a, AB \to AC, 1 > 0, 0)$ where $< a, AB \to AC, 1 > \in W_1$;
(c) $p = (\bar{a} \to a, 0, 0)$ where $a \in T$.

(Note that productions of forms (a), (b), and (c) are introuced in construction steps (2), (3), and (4), respectively.) If p has form (a), then x has form (ii). If p has form (b), then x has form (iii). Finally, if p has form (c), then x has form (i). In any of these three cases, we obtain x that has one of the required forms.

II. Assume that z has form (iii), i.e., $z =< a, AB \to AC, 1 > y$ for some $< a, AB \to AC, 1 > \in W_1, y \in ((V - \{S\}) \cup \{B'', \hat{B}, B''\})^*$, and $\#_{B''} y \leq 1$. By the inspection of $P^\sim$, we see that z can be rewritten by productions of these four forms:

(a) $(B \to B', < a, AB \to AC, 1 >, 0)$;
(b) $(B \to \hat{B}, < a, AB \to AC, 1 >, 0)$;
(c) $(\hat{B} \to B'', 0, B)$ (if $B'' \notin alph(y), i.e., \#_{B''} y = 0$);
(d) $(< a, AB \to AC, 1 > \to < a, AB \to AC, 2 >, 0, B)$ (if $B'' \notin alph(y), i.e., \#_B y = 0$).

Clearly, in cases (a) and (b), we obtain x of form (iii). If $z \Rightarrow x\ [p]$ in $G^\sim$, where p is of form (c), then $\#_{B''} x = 1$, so we get x of form (iii). Finally, if we use the production of form (d), then we obtain x of form (iv) because $\#_B z = 0$.

III. Assume that z is of form (iv), i.e., $z =< a, AB \to AC, 2 > y$, where $< a, AB \to AC, 2 > \in W_1, y \in ((V - \{S, B\}) \cup \{B', \bar{B}, B''\})^*$, and $\#_{B''} y \leq 1$. By inspection of $P^\sim$, we see that the follwoing two productions can be used to rewrite z:

(a) $(\bar{B} \to B'', 0, B'')$ $(if B'' \notin alph(y))$;
(b) $(< a, AB \to AC, 2 > \to < a, AB \to AC, 3 >, 0, \bar{B})$ $(if \bar{B} \notin alph(y))$.

In case (a), we get x of form (iv). In case (b), we have $\#_B y = 0$, so $\#_B x = 0$. Moreover, notice that $\#_{B''} x \leq 1$ in this case. Indeed, the symbol B'' can be generated only if there exists no occurrence of B'' in a given rewritten word, so no more that one occurrence of B'' appears in any sentential form. As a result, we have $\#_{B''} < a, AB \to AC, 3 > y \leq 1$, i.e., $\#_{B''} x \leq 1$. In other words, we get x of form (v).

IV. Assume that z is of form (v), i.e., $z =< a, AB \to AC, 3 > y$ for some $< a, AB \to AC, 3 > \in W_1, y \in ((V - \{S, B\}) \cup \{B'\})^* (\{[a, AB \to AC, j]; 1 \leq j \leq q + 3\} \cup \{B'', \lambda\})\ ((V - \{S, B\}) \cup \{B'\})^*$. Assume that $y = y_1 Y y_2$ with $y_1, y_2 \in ((V - \{S, B\}) \cup \{B'\})^*$. If $Y = \lambda$, then we can use no production from $P^\sim$ to rewrite z. Because $z \Rightarrow x$, we have $Y \neq \lambda$. The following cases (A) through (F) cover all possible forms of Y.

(A) Assume $Y = B''$. By inspection of $P^\sim$, we see that the only production that can rewrite z has the form $(B'' \to [a, AB \to AC, 1], < a, AB \to AC, 3 >, 0)$. In this case, we get x of form (v).

(B) Assume $Y = [a, AB \to AC, j]w, j \in \{1, \ldots, q\}$, and $f(A) \neq j$. Then z can be rewritten only according to the production $([a, AB \to AC, j] \to [a, AB \to AC, j + 1], 0, f^{-1}(j)[a, AB \to AC, j])$ (which can be used unless the rightmost symbol of $< a, AB \to AC, 3 > y_1$ is $f^{-1}(j)$). Clearly, in this case we again get x of form (v).

(C) Assume $Y = [a, AB \to AC, j], j \in \{1, \ldots, q\}, f(A) = j$. This case forms an analogy to case (B), except that the production of the form $([a, AB \to AC, f(A)] \to [a, AB \to AC, f(A) + 1], 0, 0)$ is now used.

(D) Assume $Y = [a, AB \to AC, q + 1]$. This case forms an analogy to case (B); the only change is the application of the production $([a, AB \to AC, q + 1] \to [a, AB \to AC, q + 2], 0, B'[a, AB \to AC, q + 1])$.

(E) Assume $Y = [a, AB \to AC, q + 2]$. This case forms an analogy to case (B) except that the production $([a, AB \to AC, q + 2] \to [a, AB \to AC, q + 3], 0, < a, AB \to AC, 3 > [a, AB \to AC, q + 2])$ is used.

(F) Assume $X = [a, AB \to AC, q + 3]$. By inspection of $P^{\sim}$, we see that the only production that can rewrite z is $(< a, AB \to AC, 3 > \to < a, AB \to AC, 4 > , [a, AB \to AC, q + 3], 0)$. If this production is used, we get x of form (vi).

V. Assume that z is of form (vi), i.e., $z =< a, AB \to AC, 4 > y$, where $< a, AB \to AC, 4 > \in W_1$ and $y \in ((V - \{S\}) \cup \{B'\})^*\ [a, AB \to AC, q + 3]((V - \{S\}) \cup \{B'\})^*$. By inspection of $P^{\sim}$, these two productions can rewrite z:

(a) $(B' \to B, < a, AB \to AC, 4 >, 0)$;

(b) $(< a, AB \to AC, 4 > \to < a, AB \to AC, 5 >, 0, B')$ (if $B' \notin \text{alph}(y)$).

Clearly, in case (a), we get x of form (vi). In case (b), we get x of form (vii) because $\#_{B'} y = 0$, so $y \in (V - \{S\})^*\{[a, AB \to AC, q + 3], \lambda\}(V - \{S\})^*$.

VI. Assume that z is of form (vii), i.e., $z =< a, AB \to AC, 5 > y$, where $< a, AB \to AC, 5 > \in W_1$ and $y \in (V - \{S\})^*\{[a, AB \to AC, q + 3], \lambda\}(V - \{S\})^*$. By inspection of $P^{\sim}$, one of the following two productions can be used to rewrite z:

(a) $([a, AB \to AC, q + 3] \to C, < a, AB \to AC, 5 >, 0)$;

(b) $(< a, AB \to AC, 5 > \to \bar{a}, 0, [a, AB \to AC, q + 3])$ (if $[a, AB \to AC, q + 3] \notin alph(z)$).

In case (a), we get x of form (vii). Case (b) implies $\#_{[a, AB \to AC, q+3]} y = 0$; thus, x is of form (ii).

This completes the induction step and establishes Claim 2.

Claim 3: It holds that

$$S \Rightarrow^m w \text{ in } G \text{ if and only if } S \Rightarrow^n v \text{ in } G^{\sim}$$

where $v \in g(w)$ and $w \in V^+$, for some $m, n \geq 0$.

Proof of Claim 3.

Only if: The only-if part is established by induction on m; that is, we have to demonstrate that $S \Rightarrow^m w$ in G implies $S \Rightarrow^* v$ in $G^{\sim}$ for some $v \in g(w)$ and $w \in V^+$.

Basis: Let $m = 0$. The only w is S because $S \Rightarrow^0 S$ in G. Clearly, $S \Rightarrow^0 S$ in $G^{\sim}$, and $S \in g(S)$.

Induction Hypothesis: Suppose that our claim holds for all derivations of length m or less, for some $m \geq 0$.

Induction Step: Let us consider a derivation, $S \Rightarrow^{m+1} x$, in $G, x \in V^+$. Because $m + 1 \geq 1$, there are $y \in V^+$ and $p \in P$ such that $S \Rightarrow^m y \Rightarrow x\ [p]$ in G, and

by the induction hypothesis, there is also a derivation $S \Rightarrow^n y^\sim$ in $G^\sim$ for some $y^\sim \in g(y)$. The following cases (i) through (iii) cover all possible forms of p.

(i) Let $p = S \to aA \in P$ for some $a \in T, A \in N_{CF} \cup \{\lambda\}$. Then, by Claim 1, $m = 0$, so $y = S$ and $x = aA$. By (1) in the construction of $G^\sim$, $(S \to \bar{a}A, 0, 0) \in P^\sim$. Hence, $S \Rightarrow a^\sim A$ in $G^\sim$ where $a^\sim A \in g(aA)$.

(ii) Let us assume that $p = D \to y_2 \in P, D \in N_{CF}, y_2 \in (V - \{S\}) \cup (N_{CF})^2, y = y_1 D y_3, y_1, y_3 \in V^*$ and $x = y_1 y_2 y_3$. From the definition of g, it is clear that $g(Z) = \{Z\}$ for all $Z \in N_{CF}$; therefore, we can express $y^\sim = z_1 D z_3$ where $z_1 \in g(y_1)$ and $z_3 \in g(y_3)$. Without loss of generality, we can also assume that $y_1 = ar, a \in T, r \in (V - \{S\})^*$ (see Claim 1), so $z_1 = a''r'', a'' \in g(a)$, and $r'' \in g(r)$. Moreover, by (2) in the construction, we have $(D \to y_2, \bar{a}, 0) \in P^\sim$. The following cases (a) through (e) cover all possible forms of a''.

(a) Let $a'' = \bar{a}$ (see (ii) in Claim 2). Then, we have $S \Rightarrow^n \bar{a}r''Dz_3 \Rightarrow \bar{a}r''y_2z_3\ [(D \to y_2, \bar{a}, 0)]$, and $\bar{a}r''y_2z_3 = z_1y_2z_3 \in g(y_1y_2y_3) = g(x)$.

(b) Let $a'' = a$ (see (i) in Claim 2). By (4) in the construction of $G^\sim$, we can express the derivation in $G^\sim : S \Rightarrow^n ar''Dz_3$ as $S \Rightarrow^{n-1} \bar{a}r''Dz_3 \Rightarrow ar''Dz_3\ [(\bar{a} \to a, 0, 0)]$; thus, there exists this derivation in $G^\sim : S \Rightarrow^{n-1} \bar{a}r''Dz_3 \Rightarrow \bar{a}r''y_2z_3[(D \to y_2, \bar{a}, 0)]$ with $\bar{a}r''y_2z_3 \in g(x)$.

(c) Let $a'' = < a, AB \to AC, 5 >$ for some $AB \to AC \in P$ (see (vii) in Claim 2), and let $r''Dz_3 \in (V - \{S\})^*$, i.e., $[a, AB \to AC, q+3] \notin$ alph $(r''Dz_3)$. Then, there exists this derivation in $G^\sim : S \Rightarrow^n < a, AB \to AC, 5 > r''Dz_3 \Rightarrow \bar{a}r''Dz_3\ [(< a, AB \to AC, 5 > \to \bar{a}, 0, [a, AB \to AC, q+3])] \Rightarrow \bar{a}r''y_2z_3[(D \to y_2, \bar{a}, 0)]$, and $\bar{a}r''y_2z_3 \in g(x)$.

(d) Let $a'' = < a, AB \to AC, 5 >$ (see (vii) in Claim 2). Let $[a, AB \to AC, q+3] \in$ alph $(r''Dz_3)$. Without loss of generality, we can assume that $y^\sim = < a, AB \to AC, 5 > r''Ds''[a, AB \to AC, q+3]t''$, where $s''[a, AB \to AC, q+3]t'' = z_3, sBt = y_3, s'' \in g(t), s, t \in (V - \{S\})^*$. By inspection of $P^\sim$ (see (3) in the construction of $G^\sim$), we can express the derivation in $G^\sim : S \Rightarrow^n y^\sim$ as:

$$
\begin{aligned}
S &\Rightarrow^* \bar{a}r''Ds''Bt'' \\
&\Rightarrow < a, AB \to AC, 1 > r''Ds''Bt'' \\
&\quad [(\bar{a} \to < a, AB \to AC1 >, 0, 0)] \\
&\Rightarrow^{1+|m_1m_2|} < a, AB \to AC, 1 >' Ds'\hat{B}t' \\
&\quad [m_1(B \to \hat{B}, < a, AB \to AC, 1 >, 0)m_2] \\
&\Rightarrow < a, AB \to AC, 2 > r'Ds'\bar{B}t' \\
&\quad [(< a, AB \to AC, 1 > \to < a, AB \to AC, 2 >, 0, B)] \\
&\Rightarrow < a, AB \to AC, 2 > r'Ds'B''t' \\
&\quad [\hat{B} \to B'', 0, B'')] \\
&\Rightarrow < a, AB \to AC, 3 > r'Ds'B''t' \\
&\quad [(< a, AB \to AC, 2 > \to < a, AB \to AC, 3 >, 0, \hat{B})] \\
&\Rightarrow < a, AB \to AC, 3 > r'Ds'[a, AB \to AC, 1]t' \\
&\quad [(B'' \to [a, AB \to AC, 1], < a, AB \to AC, 3 >, 0)] \\
&\Rightarrow^{q+2} < a, AB \to AC, 3 > r'Ds'[a, AB \to AC, q+3]t'
\end{aligned}
$$

$$
\begin{aligned}
& [([a, AB \to AC, 1] \to [a, AB \to AC, 2], 0, f^{-1}(1) \\
& [a, AB \to AC, 1]) \dots \\
& ([a, AB \to AC, f(A) - 1] \to [a, AB \to AC, f(A)], 0, \\
& f^{-1}(f(A) - 1)[a, AB \to AC, f(A) - 1]) \\
& ([a, AB \to ACf(A) \to [a, AB \to AC, f(A) + 1], 0, 0) \\
& ([a, AB \to AC, f(A) + 1] \to [a, AB \to AC, f(A) + 2], 0, \\
& f^{-1}(f(A) + 1)[a, AB \to AC, f(A) + 1]) \dots \\
& ([a, AB \to AC, q] \to [a, AB \to AC, q + 1], 0, \\
& f^{-1}(q)[a, AB \to AC, q]) \\
& ([a, AB \to AC, q + 1] \to [a, AB \to AC, q + 2], 0, B' \\
& [a, AB \to AC, q + 1]) \\
& ([a, AB \to AC, q + 2] \to [a, AB \to AC, q + 3], 0, \\
& < a, AB \to AC, 3 > [a, AB \to AC, q + 2])]
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \quad & < a, AB \to AC, 4 > r'Ds'[a, AB \to AC, q + 3]t' \\
& [(< a, AB \to AC, 3 > \to < a, AB \to AC, 4 >, \\
& [a, AB \to AC, q + 3], 0)] \\
\Rightarrow^{|m_3|} \quad & < a, AB \to AC, 4 > r''Ds''[a, AB \to q + 3]t'' \ [m_3] \\
\Rightarrow \quad & < a, AB \to AC, 5 > r''Ds''[a, AB \to AC, q + 3]t'' \\
& [(< a, AB \to AC, 4 > \to < a, AB \to AC, 5 >, 0, B')]
\end{aligned}
$$

where $m_1, m_2 \in \{(B \to B', < a, AB \to AC, 1 >, 0)\}^*$, $m_3 \in \{(B' \to B, < a, AB \to AC, 4 >, 0)\}^*$, $|m_3| = |m_1 m_2|, r' \in ((\text{alph}(r'') - \{B\}) \cup \{B'\})^*, g^{-1}(r) - r, s' \in ((\text{alph}(s'') - \{B\}) \cup \{B''\})^*, g^{-1}(s') = g^{-1}(s'') = s, t' \in ((\text{alph}(t'') - \{B\}) \cup \{B'\})^*, g^{-1}(t') = g^{-1}(t'') = t$.

Clearly, $\bar{a}r''Ds''Bt'' \in g(arDsBt) = g(arDy_3) = g(y)$. Thus, there exists this derivation in $G^{\sim}$: $S \Rightarrow^* \bar{a}r''Ds''Bt'' \Rightarrow \bar{a}r''y_2s''Bt'' \ [(D \to y_2, \bar{a}, 0)]$ where $z_1y_2z_3 = \bar{a}r''y_2s''Bt'' \in g(ary_2sBt) = g(y_1y_2y_3) = g(x)$.

(e) Let $a'' =< a, AB \to AC, i >$ for some $AB \to AC \in P$ and $i \in \{1, \dots, 4\}$ (see (iii) - (vi) in Claim 2). By analogy with (d), we can construct the derivation $S \Rightarrow^* \bar{a}r''Ds''Bt'' \Rightarrow \bar{a}r''y_2s''Bt'' \ [(D \to y_2, \bar{a}, 0)]$ such that $\bar{a}r''y_2s''Bt'' \in g(y_1y_2y_3) = g(x)$ (the details of this construction are left to the reader).

(iii) Let $p = AB \to AC \in P, A, C \in N_{CF}, B \in N_{CS}, y = y_1ABy_3, y_1, y_3 \in V^*, x = y_1ACy_3, y^{\sim} = z_1AYz_3, Y \in g(B), z_i \in g(y_i)$ where $i \in \{1, 3\}$. Moreover, let $y_1 = ar$ (see Claim 1), $z_1 = a''r'', a'' \in g(a), r'' \in g(r)$. The following cases (a) through (e) cover all possible forms of a''.

(a) Let $a'' = \bar{a}$. Then, by Claim 2, $Y = B$. By (3) in the construction of $G^{\sim}$, there exists the following derivation in $G^{\sim}$:

$$
\begin{aligned}
S \quad \Rightarrow^n \quad & \bar{a}r''ABz_3 \\
\Rightarrow \quad & < a, AB \to AC, 1 > r''ABu_3 \\
& [(\bar{a} \to < a, AB \to AC, 1 > 0, 0)] \\
\Rightarrow^{1+|m_1|} \quad & < a, AB \to AC, 1 > r'A\hat{B}z_3 \\
& [m_1(B \to \hat{B}, < a, AB \to AC, 1 >, 0)] \\
\Rightarrow \quad & < a, AB \to AC, 2 > r'A\hat{B}u_3
\end{aligned}
$$

$$
\begin{aligned}
& [(< a, AB \to AC, 1 > \to < a, AB \to AC, 2 >, 0, B)] \\
\Rightarrow \quad & < a, AB \to AC, 2 > r' AB'' u_3 \\
& [(\hat{B} \to B'', 0, B'')] \\
\Rightarrow \quad & < a, AB \to AC, 3 > r' AB'' u_3 \\
& [(< a, AB \to AC, 2 > \to < a, AB \to AC, 3 >, 0, \hat{B})] \\
\Rightarrow \quad & < a, AB \to AC, 3 > r' A[a, AB \to AC, 1] u_3 \\
& [(B'' \to [a, AB \to AC, 1], < a, AB \to AC, 3 >, 0)] \\
\Rightarrow^{q+2} \quad & < a, AB \to AC, 3 > r' A[a, AB \to AC, q+3] u_3 \\
& [([a, AB \to AC, 1] \to [a, AB \to AC, 2], 0, \\
& f^{-1}(1)[a, AB \to AC, 1]) \ldots \\
& ([a, AB \to AC, f(A)-1] \to [a, AB \to AC, f(A)], 0, \\
& f^{-1}(f(A)-1)[a, AB \to AC, f(A)-1]) \\
& ([a, AB \to AC, f(A)] \to [a, AB \to AC, f(A)+1], 0, 0) \\
& ([a, AB \to AC, f(A)+1]) \to [a, AB \to AC, f(A)+2], 0, \\
& f^{-1}(f(A)+1)[a, AB \to AC, f(A)+1]) \ldots \\
& ([a, AB \to AC, q] \to [a, AB \to AC, q+1], 0, \\
& f^{-1}(q)[a, AB \to AC, q]) \\
& ([a, AB \to AC, q+1] \to [a, AB \to AC, q+2], 0, B' \\
& [a, AB \to AC, q+1]) \\
& ([a, AB \to AC, q+2] \to [a, AB \to AC, q+3], 0, \\
& < a, AB \to AC, 3 > [a, AB \to AC, q+2])] \\
\Rightarrow \quad & < a, AB \to AC, 4 > r' A[a, AB \to AC, q+3] u_3 \\
& [(< a, AB \to AC, 3 > \to < a, AB \to AC, 4 >, \\
& [a, AB \to AC, q+3], 0)] \\
\Rightarrow \quad & < a, AB \to AC, 4 > r'' A[a, AB \to AC, q+3] z_3 \quad [m_2] \\
\Rightarrow \quad & < a, AB \to AC, 5 > r'' A[a, AB \to AC, q+3] z_3 \\
& [(< a, AB \to AC, 4 > \to < a, AB \to AC, 5 >, 0, B')] \\
\Rightarrow \quad & < a, AB \to AC, 5 > r'' ACz_3 \\
& [([a, AB \to AC, q+3] \to C, < a, AB \to AC, 5 >, 0)]
\end{aligned}
$$

where $m_1 \in \{(B \to B', < a, AB \to AC, 1 >, 0)\}^*, m_2 \in \{(B' \to B, < a, AB \to AC, 4 >, 0)\}^*, |m_1| = |m_2|, u_3 \in ((\text{alph}(z_3) - \{B\}) \cup \{B'\})^*, g^{-1}(u_3) = g^{-1}(z_3) = y_3, r' \in ((\text{alph}(r'') - \{B\}) \cup \{B'\})^*, g^{-1}(r') = g^{-1}(r'') = r$.

It is clear that $< a, AB \to AC, 5 > \in g(a)$; thus, $< a, AB \to AC5 > r'' ACz_3 \in g(arACy_3) = g(x)$.

(b) Let $a'' = a$. Then, by Claim 2, $Y = B$. By analogy with (ii.b) and (iii.a) in the proof of this claim (see above), we obtain: $S \Rightarrow^{n-1} \bar{a} r'' ABz_3 \Rightarrow^* < a, AB \to AC, 5 > r'' ACz_3$ so $< a, AB \to AC, 5 > r'' ACz_3 \in g(x)$.

(c) Let $a'' =< a, AB \to AC, 5 >$ for some $AB \to AC \in P$ (see (vii) in Claim 2), and let $r'' AYz_3 \in (V - \{S\})^*$. At this point, $Y = B$. By analogy with (ii.c) and (iii.a) in the proof of this claim (see above), we can construct $S \Rightarrow^{n+1}$

$\bar{a}r''ABz_3 \Rightarrow^{*} < a, AB \rightarrow AC, 5 > r''ACz_3$ so $< a, AB \rightarrow AC, 5 > r''ACz_3 \in g(x)$.

(d) Let $a'' =< a, AB \rightarrow AC, 5 >$ for some $AB \rightarrow AC \in P$ (see (vii) in Claim 2), and let $[a, AB \rightarrow AC, q+3] \in alph(r''AY_3)$. By analogy with (ii.d) and (iii.a) in the proof of this claim (see above), we can construct $S \Rightarrow^{*} \bar{a}r''ABz_3$ and, then, $S \Rightarrow^{*} \bar{a}r''ABz_3 \Rightarrow^{*} < a, AB \rightarrow AC, 5 > r''ACz_3$ so $< a, AB \rightarrow AC, 5 > r''ACz_3 \in g(arACy_3) = g(x)$.

(e) Let $a'' =< a, AB \rightarrow AC, i >$ for some $AB \rightarrow AC \in P, i \in \{i, \ldots, 4\}$, see (III) - (IV) in Claim 2. By analogy with (ii.e) and (iii.d) in the proof of this claim (see above), we can construct $S \Rightarrow^{*} \bar{a}r''ACz_3$, where $\bar{a}r''ACz_3 \in g(x)$.

If: By induction on n, we next prove that
if $S \Rightarrow^{n} v$ in $G^{\sim}$ with $v \in g(w)$ and $w \in V^{*}$ (for some $n \geq 0$),
then $S \Rightarrow^{*} w$ in G.

Basis: For $n = 0$, the only v is S as $S \Rightarrow^{0} S$ in $G^{\sim}$. Because $\{S\} = g(S)$, we have $w = S$. Clearly, $S \Rightarrow^{0} S$ in G.

Induction hypothesis: Assume the claim holds for all derivations of length n or less, for some $n \geq 0$. Let us show that it is also true for $n+1$.

Induction step: For $n+1 = 1$ (i.e. $n = 0$), there only exists a direct derivation of the form $S \Rightarrow \bar{a}A[(S \rightarrow \bar{a}A, 0, 0)]$ where $A \in N_{CF} \cup \{\lambda\}, a \in T$, and $\bar{a}A \in g(aA)$.

By (1), we have in P a production of the form $S \rightarrow aA$ and, thus, a direct derivation $S \Rightarrow aA$.

Suppose $n+1 \geq 2$ (i.e. $n \geq 1$). Consider a derivation in $G^{\sim} : S \Rightarrow^{n+1} x'$ where $x' \in g(x), x \in V^{*}$. As $n+1 \geq 2$, there exist $\bar{a} \in W_4, A \in N_{CF}, y \in V^{+}$, such that $S \Rightarrow \bar{a}A \Rightarrow^{n-1} y' \Rightarrow x'[p]$ in $G^{\sim}$, where $p \in P^{\sim}, y' \in g(y)$, and by induction hypothesis, $S \Rightarrow^{*} y$ in G.

Let us assume that $y' = z_1Zz_2, y = y_1Dy_2, z_j \in g(y_j), y_j \in (V - \{S\})^{*}, j = 1, 2$, $Z \in g(D), D \in V - \{S\}, p = (Z \rightarrow r', r_1, r_2) \in P', r_1 = 0$ or $r_2 = 0, x' = z_1r'z_2, r' \in g(r)$ for some $r \in V^{*}$ (i.e. $x' \in g(y_1ry_2)$). The following cases (i) through (iii) cover all possible forms of $y' \Rightarrow x'[p]$ in $G^{\sim}$.

(i) Let $Z \in N_{CF}$. By inspection of $P^{\sim}$, we see that $Z = D, p = (D \rightarrow r', \bar{a}, 0) \in P^{\sim}, D \rightarrow r \in P$ and $r = r'$. Thus, $S \Rightarrow^{*} y_1By_2 \Rightarrow y_1ry_2[B \rightarrow r]$ in G.

(ii) Let $r = D$. Then, by induction hypothesis, we have the derivation $S \Rightarrow^{*} y_1Dy_2$ and $y_1Dy_2 = y_1ry_2$ in G.

(iii) Let $p = ([a, AB \rightarrow AC, q+3] \rightarrow C, < a, AB \rightarrow AC, 5 >, 0), Z = [a, AB \rightarrow AC, q+3]$. Thus, $r' = C$ and $D = B \in N_{CS}$. By case (VI) in Claim 2 and the form of p, we have $z_1 =< a, AB \rightarrow AC, 5 > t$ and $y_1 = au$, where $t \in g(u), < a, AB \rightarrow AC, 5 > \in g(a), u \in (V - \{S\})^{*}$, and $a \in T$. From (3) in the construction of $G^{\sim}$, it follows that there exists a production of the from $AB \rightarrow AC \in P$. Moreover, (3) and Claim 2 imply that the derivation in $G^{\sim}$:

$$S \Rightarrow \bar{a}A \Rightarrow^{n-1} y' \Rightarrow x'[p]$$

can be expressed in the form

$$\begin{aligned} S &\Rightarrow \bar{a}A \\ &\Rightarrow^{*} \bar{a}tBz_2 \\ &\Rightarrow < a, AB \rightarrow AC, 1 > vtBz_2 \\ &\quad [(\bar{a} \rightarrow < a, AB \rightarrow AC, 1 >, 0, 0)] \end{aligned}$$

$$
\begin{aligned}
\Rightarrow^{|\theta'|} \quad & < a, AB \to AC, 1 > v\hat{B}w_2 \\
& \quad [\theta'] \\
\Rightarrow \quad & < a, AB \to AC, 1 > vB''w_2 \\
& \quad [(\hat{B} \to B'', 0, B'')] \\
\Rightarrow \quad & < a, AB \to AC, 2 > vB''w_2 \\
& \quad [(a, AB \to AC, 1 > \to < a, AB \to AC, 2 >, 0, B)] \\
\Rightarrow \quad & < a, AB \to AC, 3 > vB''w_2 \\
& \quad [(< a, AB \to AC, 2 > \to < a, AB \to AC, 3 >, 0, \hat{B})] \\
\Rightarrow \quad & < a, AB \to AC, 3 > v[a, AB \to AC, 1]w_2 \\
& \quad [(B'' \to [a, AB \to AC, 1], < a, AB \to AC, 3 >, 0)] \\
\Rightarrow^{|\theta|+2} \quad & < a, AB \to AC, 3 > v[a, AB \to AC, q+3]w_2 \\
& \quad [\theta] \\
\Rightarrow \quad & < a, AB \to AC, 4 > v[a, AB \to AC, q+3]w_2 \\
& \quad [(< a, AB \to AC, 3 > \to < a, AB \to AC, 4 >, \\
& \quad [a, AB \to AC, q+3], 0)] \\
\Rightarrow^{|\theta'|-1} \quad & < a, AB \to AC, 4 > t[a, AB \to AC, q+3]z_2 \\
& \quad [\theta''] \\
\Rightarrow \quad & < a, AB \to AC, 5 > t[a, AB \to AC, q+3]z_2 \\
& \quad [(< a, AB \to AC, 4 > \to < a, AB \to AC, 5 >, 0, B')] \\
\Rightarrow \quad & < a, AB \to AC, 5 > tCz_2 \\
& \quad [([a, AB \to AC, q+3] \to C, < a, AB \to AC, 5 >, 0)]
\end{aligned}
$$

where $\theta' \in \{(B \to B', < a, AB \to AC, 1 >, 0)\}^* \{(B \to \hat{B}, < a, AB \to AC, 1 >, 0)\}$ $\{(B \to B', < a, AB \to AC, 1 >, 0)\}^*, g(B) \cap alph(vw_2) \subseteq \{B'\}$, $g^{-1}(v) = g^{-1}(t), g^{-1}(w_2) = g^{-1}(z_2)$,
$\theta = \theta_1([a, AB \to AC, f(A)] \to [a, AB \to AC, f(A)+1], 0, 0)\theta_2([a, AB \to AC, q+1] \to [a, AB \to AC, q+2], 0, B'[a, AB \to AC, q+1])([a, AB \to AC, q+2] \to [a, AB \to AC, q+3], 0, < a, AB \to AC, 3 > [a, AB \to AC, q+2])$,
$\theta_1 = ([a, AB \to AC, 1] \to [a, AB \to AC, 2], 0, f^{-1}(1)[a, AB \to AC, 1])$
$([a, AB \to AC, 2] \to [a, AB \to AC, 3], 0, f^{-1}(2)[a, AB \to AC, 2]) \ldots$
$([a, AB \to AC, f(A)-1] \to [a, AB \to AC, f(A)], 0, f^{-1}(f(A)-1)[a, AB \to AC, f(A)-1])$,
where $f(A)$ implies $q_1 = \lambda$,
$\theta_2 = ([a, AB \to AC, f(A)+1] \to [a, AB \to AC, f(A)+2], 0, f^{-1}(f(A)+1)\,[a, AB \to AC, f(A)+1]) \ldots ([a, AB \to AC, q] \to [a, AB \to AC, q+1], 0, f^{-1}(q)[a, AB \to AC, q])$, where $f(A) = q$ implies $q_2 = \lambda, \theta'' \in .\{(B' \to B, < a, AB \to AC, 4 >, 0)\}^*$.

The above derivation implies that the rightmost symbol of t must be A. As $t \in g(u)$, the rightmost symbol of u must be A as well. That is, $t = s'A, u = sA$ and $s' \in g(s)$ (for some $s \in (V - \{S\})^*$). By the induction hypothesis, there exists a derivation in $G : S \Rightarrow^* asABy_2$. Because $AB \to AC \in P$, we get $S \Rightarrow^* asABy_2 \Rightarrow asACy_2[AB \to AC]$, where $asACy_2 = y_1 r y_2$.

By (i), (ii), (iii) and inspection of $P^{\sim}$, we see we have considered all possible derivations of the form $S \Rightarrow^{n+1} x'$ (in $G^{\sim}$) , so we have established Claim 3 by the principle of induction.

The equivalence of G and $G^{\sim}$ can be easily derived from Claim 3. By the definition of g, we have $g(a) = \{a\}$ for all $a \in T$. Thus, by Claim 3, we have for all $x \in T^*$:

$$S \Rightarrow^* x \text{ in } G \text{ if and only if } S \Rightarrow^* x \text{ in } G^{\sim}$$

Consequently, $L(G) = L(G^{\sim})$. We conclude that

$$\mathbf{CS} = \mathbf{prop} - \mathbf{SSC}(1,2)$$

and the theorem holds. Q.E.D.

Corollary 8
$\mathbf{CS} = \mathbf{prop} - \mathbf{SSC}(1,2) = \mathbf{prop} - \mathbf{SSC} = \mathbf{prop} - \mathbf{SC}(1,2) = \mathbf{prop} - \mathbf{SC}.$

We now turn to the investigation of *ssc*-grammars of degree (1,2) with erasing productions.

Theorem 9 $\mathbf{RE} = \mathbf{SSC}(1,2)$.

Proof. Clearly, we have the containment $\mathbf{SSC}(1,2) \subseteq \mathbf{RE}$; hence, it suffices to show $\mathbf{RE} \subseteq \mathbf{SSC}(1,2)$. Every language $L \in \mathbf{RE}$ can be generated by a grammar $G = (V, T, P, S)$ in which each production is of the form $AB \rightarrow AC$ or $A \rightarrow x$, where $A, B, C \in V - T, x \in \{\lambda\} \cup T \cup (V - T)^2$ (see [2]). Thus, the containment $\mathbf{RE} \subseteq \mathbf{SSC}(1,2)$ can be established by analogy with the proof of Theorem 7 (the details are left to the reader) Q.E.D.

Corollary 10 $\mathbf{RE} = \mathbf{SSC}(1,2) = \mathbf{SSC} = \mathbf{SC}(1,2) = \mathbf{SC}.$

Corollaries 2,4, 8, and 11 imply the main result of this paper:

Corollary 11

$$\mathbf{CF}$$
$$\subset$$
$$\mathbf{prop} - \mathbf{SSC} = \mathbf{prop} - \mathbf{SSC}(2,1) = \mathbf{prop} - \mathbf{SSC}(1,2) =$$
$$\mathbf{prop} - \mathbf{SC} = \mathbf{prop} - \mathbf{SC}(2,1) = \mathbf{prop} - \mathbf{SC}(1,2) = \mathbf{CS}$$
$$\subset$$
$$\mathbf{SSC} = \mathbf{SSC}(2,1) = \mathbf{SSC}(1,2) = \mathbf{SC} = \mathbf{SC}(2,1) = \mathbf{SC}(1,2) = \mathbf{RE}$$

Acknowledgement: The authors are indebted to the anonymous referee for many suggestions concerning the first version of this paper.

References

[1] Paun, Gh. (1985): A Variant of Random Context Grammars: Semi-Conditional Grammars. Theoretical Computer Science 41, 1-17.

[2] Penttonen, M. (1974): One-Sided and Two-Sided Context in Formal Grammars. Inform. Contr. 25, 371-392.

[3] Harrison, M. (1979): Introduction to Formal Language Theory. Addition Wesley, Reading (Mass.).

Received August 27, 1993

Acta Cybernetica, Vol. 11, No. 4, Szeged, 1994

Invariance groups of threshold functions

E. K. Horváth *

Permutations of variables leaving a given Boolean functien $f(x_1, \ldots, x_n)$ invariant form a group, which we call the *invariance group* G of the function. We obtain that for threshold functions G is isomorphic to a direct product of symmetric groups.

A *threshold function* is a Boolean function, i.e. a mapping $\{0,1\}^n \to \{0,1\}$ with the following property: There exist real numbers $w_1, \ldots, w_n$, t such that

$$f(x_1, \ldots, x_n) = 1 \text{ iff } \sum_{i=1}^{n} w_i x_i \geq t,$$

where w_i is called the *weight* of x_i for $i = 1, 2, \ldots, n$, and t is a constant called the *threshold value*.We can suppose without loss of generality that

$$w_1 < w_2 < \ldots < w_n. \qquad [1], [2]$$

Throughout this paper, we use the notation: $(X) = (x_1, \ldots, x_n)$; $W = (w_1, \ldots, w_n)$; $W(X) = \sum_{i=1}^{n} w_i x_i$. Let X stand for the set consisting of the symbols $x_1, \ldots, x_n$. We define an ordering on the set X in the following way: $x_i < x_j$ iff $w_i < w_j$. For any permutation π of X, the *moving set* of π, denoted by $M(\pi)$, consists of all elements x of X satisfying $\pi(x) \neq x$. Denote by S_X the group of all permutations of the set X, and by S_k the symmetric group of degree k. If $P = (p_1, \ldots, p_n) \in \{0,1\}^n$ and $\sigma \in S_X$, then let $\sigma(P) = (\sigma(p_1), \ldots, \sigma(p_n))$ and $\sigma(X) = (\sigma(x_1), \ldots, \sigma(x_n))$.

Let $(X; \leq)$ be an ordered set. Consider a partition C of X. As usual, we shall denote the class of C that contains $x \in X$ by $\bar{x}$. We call C *convex* if $x_i \leq x_j \leq x_k$ and $\bar{x}_i = \bar{x}_k$ together imply $\bar{x}_i = \bar{x}_j$. For any convex partition C of X, the ordering of X induces an ordering of the set of blocks of C in a natural way: $\bar{x}_i \leq \bar{x}_j$ iff $x_i \leq x_j$.

Theorem 1 *For every n-ary threshold function f there exists a partition C_f of X such that the invariance group G of f consists of exactly those permutations of S_X which preserve each block of C_f .*

Conversely, for every partition C of X there exists a threshold function f_C such that the invariance group G of f_C consists of exactly those permutations of S_X that preserve each block of C.

Proof. First, consider an arbitrary n-ary threshold function f. Let us define the relation $\sim$ on the set X as follows: $i \sim j$ iff $i = j$ or f is invariant under the transposition $(x_i x_j)$. Clearly, this relation is reflexive, and symmetric. Moreover, it is transitive because

*JATE, Bolyai Intézet, Aradi Vértanúk Tere 1, H-6720 Szeged, Hungary e-mail H7753Kat@HUELLA.BITNET

$$(x_i x_j)(x_j x_k)(x_i x_j) = (x_i x_k).$$

Hence $\sim$ is an equivalence relation.

Claim 1. The partition C_f defined by $\sim$ is convex.

Proof. If it is not so then there exist a Boolean vector $D = (d_1, \ldots, d_n) \in \{0,1\}^n$ and $1 \leq i \leq j \leq k \leq n$ with $x_i \sim x_k$ such that

$$d + w_i d_j + w_j d_i + w_k d_k < t, \tag{1}$$

$$d + w_i d_i + w_j d_j + w_k d_k \geq t, \tag{2}$$

if $d = \sum_{q \neq i,j,k} c_q d_q$. Now (1) and (2) imply $d_i = 0$, $d_j = 1$. Since $x_i \sim x_k$, from (1) and (2) we infer:

$$d + w_i d_k + w_j d_i + w_k d_j < t, \tag{3}$$

$$d + w_i d_k + w_j d_j + w_k d_i \geq t. \tag{4}$$

Assume $d_k = 0$. Then $d + w_k < t \leq d + w_j$ by (3) and (2), whence $w_k < w_j$, which is a contradiction. On the other hand, suppose $d_k = 1$. Then because of (1) and (4), $d + w_i + w_k < t \leq d + w_i + w_j$, which is also a contradiction.

For the reason of convexity, the blocks of $\sim$ may be given this way:

$$\begin{aligned} C_1 &= \{x_1, \ldots, x_{i_1}\}, \\ C_2 &= \{x_{i_1+1}, \ldots, x_{i_1+i_2}\}, \\ C_l &= \{x_{i_1+i_2+\ldots+i_{l-1}+1}, \ldots, x_{i_1+\ldots+i_l}\}. \end{aligned} \tag{5}$$

Every permutation that is a product of some "permitted" transpositions preserves the blocks of C_f, and belongs to G. We show that if a permutation does not preserve each blocks of C_f defined by $\sim$, then it cannot belong to G.

Lemma 1 *Let* $\gamma = (x_{j_1} x_{j_2} \ldots x_{j_{k-1}} y x_{j_k} \ldots x_{j_m}) \in S_X$ *be a cycle of length* $m+1$ *with* $x_{j_s} \in C_p$, $1 \leq s \leq m$, $y \in C_q$, $p \neq q$. *Then* $\gamma \notin G$.

Proof. Let us confine our attention to the following:

$$(y x_{j_{k-1}})(x_{j_1} x_{j_2} \ldots x_{j_{k-1}} y x_{j_k} \ldots x_{j_m}) = (x_{j_1} x_{j_2} \ldots x_{j_m})(y),$$

so

$$(y x_{j_{k-1}}) = (x_{j_1} x_{j_2} \ldots x_{j_m})(x_{j_1} x_{j_2} \ldots x_{j_{k-1}} y x_{j_k} \ldots x_{j_m})^{-1}.$$

If γ were an element of G, then $(y x_{j_{k-1}})$ would be also an element of G, which contradicts the definition of $\sim$.

Claim 1. If a cycle $\beta \in S_X$ has entries from at least two blocks of C_f, then $\beta \notin G$.

Proof. Given the convex partition C_f of $(X;\leq)$, for any cycle β of length k we construct a sequence of cycles of increasing length, called the *downward sequence* of β, as follows: Let $\bar{x}_p$, $\bar{x}_q$ $(\bar{x}_p > \bar{x}_q)$ the two greatest blocks of C_f for which x_p, x_q are entries of β. We cancel some entries of β in such a way that we keep *all* entries in $\bar{x}_p$ *and* the greatest entry in $\bar{x}_q$, and we delete all the remaining entries of β. This results in the initial cycle of the downward sequence $\beta_{(r)}$ of length r; $r \geq 2$. We do not need to define members of the downward sequence with subscripts less then r. If we have constructed $\beta_{(i)}$, we obtain the next member $\beta_{(i+1)}$ of the downward sequence by taking back the greatest cancelled (and not restored yet) entry of β in its original place. Thus, the final member of the downward sequence is $\beta_{(k)} = \beta$. Let us denote by $x^{[i]}$ $(i > r)$, the "new" entry of $\beta_{(i)}$. If $i \leq r$, then we do not have to define $x^{[i]}$. As an illustration take the following:

$$X = \{x_1, \ldots, x_8\},$$

$$\begin{aligned} C_1 &= \{x_1, x_2\}, \\ C_2 &= \{x_3, x_4\}, \\ C_3 &= \{x_5, x_6, x_7\}, \\ C_4 &= \{x_8\}, \end{aligned}$$

and

$$\beta = (x_4x_5x_1x_7x_3) = (x_1x_7x_3x_4x_5).$$

The downward sequence is:

$$\beta_{(3)} = (x_7x_4x_5),$$

$$\beta_{(4)} = (x_7x_3x_4x_5), \qquad x^{[4]} = x_3,$$

$$\beta_{(5)}(= \beta) = (x_1x_7x_3x_4x_5), \qquad x^{[5]} = x_1.$$

It is obvious from the construction of the downward sequence that the weight of an arbitrary variable occuring in $\beta_{(i)}$ is not smaller than the weight of $x^{[i+1]}$. By Lemma 1, the initial cycle of the downward sequence (in our example $\beta_{(3)}$) is not in G. In order to prove that $\beta \notin G$, we show that if there exist $A_{(i)} = (a_{(i),1}, \ldots, a_{(i),n})$ and $B_{(i)} = (b_{(i),1}, \ldots, b_{(i),n})$ with $A_{(i)}, B_{(i)} \in \{0,1\}^n$ such that $f(A_{(i)}) = 0$ and $f(B_{(i)}) = 1$ and $\beta_{(i)}(A_{(i)}) = B_{(i)}$, then we are able to construct $A_{(i+1)} = (a_{(i+1),1}, \ldots, a_{(i+1),n})$ and $B_{(i+1)} = (b_{(i+1),1}, \ldots, b_{(i+1),n})$ with $A_{(i+1)}$, $B_{(i+1)} \in \{0,1\}^n$ satisfying $f(A_{(i+1)}) = 0$ and $f(B_{(i+1)}) = 1$ and $\beta_{(i+1)}(A_{(i+1)}) = B_{(i+1)}$. Let us denote with superscripts $[l(j)]$, and $[r(j)]$ the left, and the right neighbour of $x^{[j]}$ in the cycle $\beta_{(j)}$, respectively. In our example: $x^{[l(5)]} = x_5$, $x^{[r(5)]} = x_7$ because $x^{[5]} = x_1$. (For the sake of clarity: $[r([l(j)])] = [l([r(j)])] = j$; moreover, $x^{[r(j)]}$ and $x^{[j]}$ are the images of $x^{[j]}$ and $x^{[l(j)]}$, respectively.) We shall use this notation for the corresponding components of a concrete Boolean vector as well, i.e. for example: $a_{(i)}^{[l(j)]}$ and $a_{(i)}^{[r(j)]}$. We have four possibilities for $A_{(i)}$:

Case 1. $a_{(i)}^{[i+1]} = 0$, $a_{(i)}^{[r(i+1)]} = 0$.

Case 2. $a_{(i)}^{[i+1]} = 1$, $a_{(i)}^{[r(i+1)]} = 1$.

Case 3. $a_{(i)}^{[i+1]} = 1$, $a_{(i)}^{[r(i+1)]} = 0$.

Case 4. $a_{(i)}^{[i+1]} = 0$, $a_{(i)}^{[r(i+1)]} = 1$.

We show that in the first three cases A_i is appropriate for A_{i+1}. In Case 4 the only thing we have to do is to transpose two components of A_i in order to get a suitable A_{i+1}.

Case 1. $a_{(i)}^{[i+1]} = 0$, $a_{(i)}^{[r(i+1)]} = 0$.

Even though β_{i+1} bypasses $x^{[i+1]}$, $\beta_{(i+1)}(A_{(i)}) = \beta_{(i)}(A_{(i)})$ holds because $a_{(i)}^{[i+1]} = a_{(i)}^{[r(i+1)]}$. If $A_{(i+1)} = A_{(i)}$, then $\beta_{(i+1)}(A_{(i+1)}) = \beta_{(i)}(A_{(i)}) = B_{(i)}$. So let us choose $B_{(i+1)} = B_{(i)}$. Thus $f(A_{(i+1)}) = 0$, $f(B_{(i+1)}) = 1$, and $\beta_{(i+1)}(A_{(i+1)}) = B_{(i+1)}$ are satisfied.

	$x^{[l(i+1)]}$	$x^{[i+1]}$	$x^{[r(i+1)]}$
$A_{(i)}$	$a_{(i)}^{[l(i+1)]}$	0	0
$B_{(i)}$	0	0	$b_{(i)}^{[r(i+1)]}$
$A_{(i+1)}$	$a_{(i+1)}^{[l(i+1)]}$	0	0
$B_{(i+1)}$	0	0	$b_{(i+1)}^{[r(i+1)]}$

Case 2. $a_{(i)}^{[i+1]} = 1$, $a_{(i)}^{[r(i+1)]} = 1$.

The situation is the same as in Case 1: $a_{(i)}^{[i+1]} = a_{(i)}^{[r(i+1)]}$. Let $A_{(i+1)} = A_{(i)}$. Then $\beta_{(i+1)}(A_{(i+1)}) = \beta_{(i)}(A_{(i)}) = B_{(i)}$, hence let us choose $B_{(i+1)} = B_{(i)}$. Thus $f(A_{(i+1)}) = 0$, $f(B_{(i+1)}) = 1$, and $\beta_{(i+1)}(A_{(i+1)}) = B_{(i+1)}$ are satisfied for the reason as in Case 1.

	$x^{[l(i+1)]}$	$x^{[i+1]}$	$x^{[r(i+1)]}$
$A_{(i)}$	$a_{(i)}^{[l(i+1)]}$	1	1
$B_{(i)}$	1	1	$b_{(i)}^{[r(i+1)]}$
$A_{(i+1)}$	$a_{(i+1)}^{[l(i+1)]}$	1	1
$B_{(i+1)}$	1	1	$b_{(i+1)}^{[r(i+1)]}$

Case 3. $a_{(i)}^{[i+1]} = 1$, $a_{(i)}^{[r(i+1)]} = 0$.

Now, $A_{(i)}$ is appropriate for $A_{(i+1)}$ but we cannot guarantee the same for $B_{(i)}$ and $B_{(i+1)}$. Let $A_{(i+1)} = A_{(i)}$, and $B_{(i+1)} = \beta_{(i+1)}(A_{(i+1)})$. We can get the Boolean vector $B_{(i+1)}$ from $B_{(i)}$ if we transpose $b_{(i)}^{[i+1]}$ and $b_{(i)}^{[l(i+1)]}$, i.e.:

$$b_{(i+1)}^{[l(i+1)]} = 1, \text{ and } b_{(i+1)}^{[i+1]} = 0,$$

while

$$b_{(i)}^{[l(i+1)]} = 0, \text{ and } b_{(i)}^{[i+1]} = 1;$$

furthermore, all the other components of $B_{(i+1)}$ and $B_{(i)}$ are identical. Since $x^{[i+1]}$ has the smallest weight in $\beta_{(i+1)}$, we get

$$\sum_{j=1}^{n} w_j b_{(i),j} \leq \sum_{j=1}^{n} w_j b_{(i+1),j},$$

which means that $f(B_{(i+1)}) = 1$. Moreover, $f(A_{(i+1)}) = 0$, and $\beta_{(i+1)}(A_{(i+1)}) = B_{(i+1)}$ are satisfied.

	$x^{[l(i+1)]}$	$x^{[i+1]}$	$x^{[r(i+1)]}$
$A_{(i)}$	$a_{(i)}^{[l(i+1)]}$	1	0
$B_{(i)}$	0	1	$b_{(i)}^{[r(i+1)]}$
$A_{(i+1)}$	$a_{(i+1)}^{[l(i+1)]}$	1	0
$B_{(i+1)}$	1	0	$b_{(i+1)}^{[r(i+1)]}$

Case 4. $a_{(i)}^{[i+1]} = 0$, $a_{(i)}^{[r(i+1)]} = 1$.

Let us construct $A_{(i+1)}$ from $A_{(i)}$ as follows: Put $a_{(i+1)}^{[i+1]} = 1$, $a_{(i+1)}^{[r(i+1)]} = 0$, $a_{(i+1),j} = a_{(i),j}$ if $a_{(i+1),j} \neq a_{(i+1)}^{[i+1]}$ or $a_{(i+1),j} \neq a_{(i+1)}^{[r(i+1)]}$. (Transpose $a_{(i)}^{[i+1]}$ and $a_{(i)}^{[r(i+1)]}$ in the Boolean vector $A_{(i)}$ (and keep all the other components of it unchanged) to get $A_{(i+1)}$.) Since $x^{[i+1]}$ has the smallest weight in $\beta_{(i+1)}$, we get

$$\sum_{j=1}^{n} w_j a_{(i+1),j} \leq \sum_{j=1}^{n} w_j a_{(i),j};$$

hence $f(A_{(i+1)}) = 0$. Let $B_{(i+1)} = \beta_{(i+1)}(A_{(i+1)})$. With this choice $B_{(i+1)} = B_i$, hence $f(B_{(i+1)}) = 1$.

	$x^{[l(i+1)]}$	$x^{[i+1]}$	$x^{[r(i+1)]}$
$A_{(i)}$	$a_{(i)}^{[l(i+1)]}$	0	1
$B_{(i)}$	1	0	$b_{(i)}^{[r(i+1)]}$
$A_{(i+1)}$	$a_{(i+1)}^{[l(i+1)]}$	1	0
$B_{(i+1)}$	1	0	$b_{(i+1)}^{[r(i+1)]}$

Claim 2 is proved.

Every permutation that is a product of disjoint cycles such that any of them preserves each blocks of C_f belongs to the invariance group G of f. We have to show, that if not all of the factors have this property, then the permutation does not leave the threshold function f invariant.

Lemma 2 *Let $\pi \in S_X$ of the form $\pi = \pi_2\pi_1$, where $\pi_1, \pi_2 \in S_X$, with $M(\pi_1) \cap M(\pi_2) = \emptyset$ and $\pi_1 \notin G$. Then $\pi \notin G$.*

Proof. Suppose that it is not so, i.e. $\pi \in G$. Now $\pi_1 \notin G$ means that there exist $X_0, X_1 \in \{0,1\}^n$ with $f(X_0) = 0, f(X_1) = 1$, and $\pi_1(X_0) = X_1$. Let $X_2 = \pi_2(X_1)$, i.e. $X_2 = \pi(X_0)$. Since $f(X_2) = 1$ contradicts the assumption $\pi \in G$, we infer $f(X_2) = 0$. Let $X_3 = \pi_1(X_2)$. As $M(\pi_1) \cap M(\pi_2) = \emptyset$, we have $\pi_1\pi_2 = \pi_2\pi_1$. Therefore $X_3 = \pi(X_1)$. The assumption $\pi \in G$ implies $f(X_3) = 1$. Looking at the infinite series of Boolean vectors

$$X_0, X_1, \ldots, X_n, \ldots$$

we can establish in the same way that if $i = 2k$, $k \in \mathbf{N}$, then $f(X_i) = 0$, while if $i = 2k+1$ then $f(X_i) = 1$. On the other hand,

$$W(X) = S(X)^{[1]} + S(X)^{[2]} + S(X)^{[3]},$$

where $S(X)^{[1]} = \sum_{x_j \in M(\pi_1)} w_j x_j$, $S(X)^{[2]} = \sum_{x_j \in M(\pi_2)} w_j x_j$, $S(X)^{[3]} = \sum_{x_j \notin M(\pi)} w_j x_j$. With this notation: $S(X_0)^{[1]} < S(X_1)^{[1]}$, $S(X_0)^{[2]} = S(X_1)^{[2]}$, $S(X_0)^{[3]} = S(X_1)^{[3]}$. For the series of $S(X_i)^{[1]}$:

$$S(X_0)^{[1]} < S(X_1)^{[1]} = S(X_2)^{[1]} < S(X_3)^{[1]} = S(X_4)^{[1]} < \ldots, \tag{6}$$

as applying π_2 changes only $S(X_i)^{[2]}$; moreover, $f(X_{2k}) = 0$ and $f(X_{2k+1}) = 1$ imply $W(X_{2k}) < W(X_{2k+1})$, hence $S(X_{2k})^{[1]} < S(X_{2k+1})^{[1]}$. On the other hand, if z is the order of π_1, then $S(X_0)^{[1]} = S(X_{2z})^{[1]}$, which contradicts (6).

Claim 3. For $\pi \in S_X$, let $\pi = \gamma_1 \ldots \gamma_r$ where γ_i are disjoint cycles. If there exists a γ_j with $1 \leq j \leq r$ and $\gamma_j \notin G$, then $\pi \notin G$.

Proof. It is easy to see if there is only one such γ_j. If there is more, then $\pi \notin G$ is an immediate consequence of Lemma 2.

Claim 1, Claim 2, and Claim 3 together provide a proof of the first part of the Theorem.

For proving the converse of the theorem, we show first that for any n there exist a n-ary threshold function which is *rigid* in the sense that its invariance group has only one element (the identity permutation).

Suppose n is odd. With $n = 2k+1$, consider the following weights:

w_1	w_2	$\ldots$	w_k	w_{k+1}	w_{k+2}	$\ldots$	w_{2k}	w_{2k+1}
$-k$	$-k+1$	$\ldots$	-1	0	1	$\ldots$	$k-1$	k

(7)

Let $t = 0$. We prove that for any transposition τ of form $(x_j x_{j-1})$ where $2 \leq j \leq n$ there exists a Boolean vector $U = (u_1, \ldots, u_n) \in \{0,1\}^n$ such that $f(U) = 1$ and

$f(\tau(U)) = 0$. For a fixed j let $u_j = 1$, $u_{n+1-j} = 1$, $u_i = 0$ if $i \neq j$, $i \neq n+1-j$. It is obvious that $f(U) = 1$; however, $f(\tau(U)) = 0$. Hence f is rigid.

If $n = 2k$, then the weights can be chosen as

w_1	w_2	...	w_{k-1}	w_k	w_{k+1}	w_{k+2}	...	w_{2k-1}	w_{2k}
$-k$	$-k+1$	...	-2	-1	1	2	...	$k-1$	k

(8)

Let $t = 0$. The method is almost the same as before, i.e. consider the following $U = (u_1, \ldots, u_n)$.: If $j \neq k+1$ then let $u_j = 1$, $u_{n+1-j} = 1$, $u_i = 0$ if $i \neq j, -j$. If $j = k+1$ then let $u_{k+1} = 1$ and $u_i = 0$ if $i \neq k+1$. If $\tau = (x_j x_{j-1})$, where $2 \leq j \leq n$, then $f(U) = 1$ while $f(\tau(U)) = 0$.

Now, we construct a threshold function g_C for an arbitrary partition C of an arbitrary ordered set X of variables. Denote now by $\sim^*$ the equivalence relation on X defined by C. First, suppose that C is convex. Let $i_1, \ldots, i_l$ denote the number of elements of the blocks of C, respectively. Consider the rigid function f of l variables that is defined in (7) or (8), depending on the parity of l. Take the weight w_1 i_1 times, the weight w_2 i_2 times and so on in order to define a threshold function g of $n = i_1 + i_2 + \ldots + i_l$ variables. Variables of g with the same weight are permutable. However, transpositions σ of form $(x_j x_{j-1})$, where $2 \leq j \leq n$ and $j \not\sim^* j-1$, are "forbidden" for g because if we consider the corresponding U and construct a Boolean vector V of dimension n from U by rewriting it in the following way: instead of u_m $(m = 1, \ldots, l)$, write 0 i_m times, whenever $u_m = 0$; and write 1 (once) then 0 $i_m - 1$ times otherwise; then we shall get a Boolean vector V of dimension n, for which $g(V) = 1$ while $g(\sigma(V) = 0$. If C is not convex, the only thing we have to do is to reindex the variables in order to get a convex partition. After constructing a threshold function for the rearranged variables with the procedure described above, put the original indexes back and the desired threshold function is ready. Theorem is proved.

The invariance group G_B of an arbitrary Boolean function is not necessarily of the form

$$G_B \cong S_{i_1} \times \ldots \times S_{i_l}. \tag{9}$$

For example, let h be the following: $h(x_1, \ldots, x_n) = 1$ iff there exists i such that $x_i = 1$, $x_{i\oplus 1} = 1$, $x_j = 0$ if $j \neq i, i+1$ where $\oplus$ means addition *mod* n. The invariance group of h contains the cycle $(x_1, \ldots, x_n)$ and its powers but it does not contain transpositions of form $(x_i x_{i+1})$.

However, there exist Boolean functions with invariance groups of the form (9), which are not threshold functions.

Permutable variables of a threshold function does not mean equal weights. Here is an example: $h(x) = x_1x_2x_4 \vee x_3x_4$. This is a threshold function with the following weights, and threshold value:

w_1	w_2	w_3	w_4	t
1	2	3	4	7

The transposition $(x_1 x_2)$ is "permitted" but the others are not.

But the weights can always be chosen to be identical for variables belonging to the same equivalence class. If the j-th class $C_j = \{x_{i_1+i_2+\ldots+i_{j-1}+1}, \ldots, x_{i_1+\ldots+i_j}\}$ by te notation of (5), then let $w_{[j]} = \frac{w_{i_1+i_2+\ldots+i_{j-1}+1}+\ldots+w_{i_1+\ldots+i_j}}{i_j}$. Replace $w_{i_1+i_2+\ldots+i_{j-1}+1}, \ldots, w_{i_1+\ldots+i_j}$, by $w_{[j]}$. Since $x_{i_1+i_2+\ldots+i_{j-1}+1}, \ldots, x_{i_1+\ldots+i_j}$

are from the same equivalence class, for fixed $x_1, \ldots, x_{i_1+\ldots i_{j-1}}$ and $x_{i_1+\ldots i_j+1}, \ldots x_{i_1+\ldots i_l}$, the fact that $W(X)$ exceeds t (or not) depends only on the number r of 1-s among the coordinates $x_{i_1+i_2+\ldots+i_{j-1}+1}, \ldots, x_{i_1+\ldots+i_j}$; moreover, $W(X)$ has a maximum (minimum) if we put all our 1-s to places with the greatest (smallest) weights possible. Obviously

$$\frac{w_{i_1+\ldots i_{r-1}+1} + \ldots + w_{i_1+\ldots i_{j-1}+1+r}}{r} \leq w_{[j]};$$

moreover,

$$w_{[j]} \leq \frac{w_{i_1+\ldots i_j-r} + \ldots + w_{i_1+\ldots i_j}}{r}.$$

Hence

$$w_{i_1+\ldots i_{r-1}+1} + \ldots + w_{i_1+\ldots i_{j-1}+1+r} \leq r w_{[j]} \leq w_{i_1+\ldots i_j-r} + \ldots + w_{i_1+\ldots i_j}.$$

Consequently, after replacing $w_{i_1+i_2+\ldots+i_{j-1}+1}, \ldots, w_{i_1+\ldots+i_j}$ by $w_{[j]}$, we still have the same threshold function.

Acknowledgement I would like to thank B. Csákány for raising the problem, for discussions, and his helpful suggestions.

References

[1] Algebraic aspects of threshold logic Russian Kibernetika (Kiev), p. 26–30, 1980

[2] N. N. Aĭzenberg, A. A. Bovdi, È. I. Gergo , F. È. Geche English transl. in Cybernetics vol. 16, 1980 p. 188–193

[3] Béla Bódi, Elemér Hergő, Ferenc Gecseg A lower bound of the number of threshold functions Trans.IEEE,EC-14 6, 1965, p. 926–929

[4] S. Yajima and T. Ibaraki Threshold logic

[5] H.G. Booker and N. DeClaris Academic Press, London and New York Ching Lai Sheng 1969

Received March 26, 1994

Acta Cybernetica, Vol. 11, No. 4, Szeged, 1994

On the Complexity of Dynamic Tests for Logic Functions

V.A.Vardanian*

Abstract

A generalisation of the concept of *dynamic test* is proposed for detecting logic and parametric faults at input / output terminals of logic networks realising $k-$ valued logic functions ($k \geq 2$). Upper and lower bounds on the *complexity* (i.e., length) of minimal dynamic tests are obtained for various classes of logic functions.

1 Introduction

In dynamic testing of combinational logic networks (see [1,2]) the fault-free and faulty circuits are distinguished if they have different dynamic (i.e., time varying) behaviors (output level variations) under the same input stimulation by a transition signal. It should be noted (see [1-3]) that there are statically undetectable logic faults, as well as parametric faults (e.g., inadmissible variations of the magnitude of time delays), which may become detectable only in dynamic testing. In [3] a notion of dynamic test was introduced for input/output terminals (I/O faults) of combinational networks since in many cases faults are more likely to occur at the input/output terminals rather than inside. The dynamic test [3,4] is defined to be a set of input patterns sensitising the output of the network with respect to simultaneous switching of every feasible subset of input variables. Evidently, the dynamic test for I/O faults does not depend on the internal structure of the network, but depends only on the function realised by the output. In [3-7] some classes of logic and parametric I/O faults are described to be detectable by dynamic tests, and the complexity (i.e., length) of minimal dynamic tests is investigated for various classes of logic functions.

In this paper, a generalization of the notion of dynamic test called (dynamic) test of regularity, is proposed for $k-$ valued logic functions, $k \geq 2$. As a result, the class of detectable I/O faults is considerably enlarged. A notion of stability dual to that of sensitivity is introduced, and the test of regularity is defined to be a set of input patterns that are sufficient to both sensitize and stabilize the logic function with respect to simultaneous switching of every feasible subset of input variables. Upper and lower bounds on the complexity of minimal tests of regularity are obtained for some classes of k-valued logic functions.

*Institute of Informatics and Automation Problems of the Armenian National Academy of Sciences, Yerevan, Armenia

2 Notations and Definitions

Let $E_k = \{0, 1, ..., k-1\}, k \geq 2$, denote

$$E_k^n = \{\tilde{\alpha}/\tilde{\alpha} = (\alpha_1, ..., \alpha_n), \alpha_i \in E_k, i = \overline{1,n}\};$$

$$P_k(n) = \{f/f : E_k^n \mapsto E_k\};$$

$$G_I(\tilde{\alpha}) = \{\tilde{\beta}/\tilde{\beta} \in E_k^n, (\beta_j \neq \alpha_j) \leftrightarrow (j \in I)\}$$

where $I \subseteq N_n = \{1, 2, ..., n\}, I \neq \emptyset, \tilde{\alpha} \in E_k^n$.

For $k \geq 2, \tilde{\alpha} \in E_k^n, I = \{i_1, ..., i_s\} \subseteq N_n, I \neq \emptyset$, the set $G_I(\tilde{\alpha}) \bigcap E_2^n$ has only one element denoted in the sequel by

$$\tilde{\alpha}^I = (\alpha_1, ..., \alpha_{i_1-1}, \overline{\alpha}_{i_1}, \alpha_{i_1+1}, ..., \alpha_{i_s-1}, \overline{\alpha}_{i_s}, \alpha_{i_s+1}, ..., \alpha_n)$$

where $\overline{\sigma} = 1 - \sigma, \sigma \in E_2$.

Definition 2.1 . The function $f(x_1, ..., x_n) \in P_k(n)$ is *sensitive (stable)* at vector $\tilde{\alpha} \in E_k^n$ with respect to the subset of variables $\{x_{i_1}, ..., x_{i_s}\} \subseteq \{x_1, ..., x_n\}$ if there exists a vector $\tilde{\beta} \in G_{\{i_1,...,i_s\}}(\tilde{\alpha})$ such that $f(\tilde{\alpha}) \neq f(\tilde{\beta})$ (respectively , $f(\tilde{\alpha}) = f(\tilde{\beta})$). The function f is sensitive (stable) with respect to $\{x_{i_1}, ..., x_{i_s}\}$ if there exists a vector $\tilde{\alpha} \in E_k^n$ at which f is sensitive (stable) with respect to $\{x_{i_1}, ..., x_{i_s}\}$.

Definition 2.2. The function $f \in P_k(n)$ is said to be *regular* if it is both sensitive and stable with respect to every nonempty subset of variables $\{x_{i_1}, ..., x_{i_s}\} \subseteq \{x_1, ..., x_n\}$.

Denote by $R_k(n)$ the set of all regular functions $f \in P_k(n)$.

We shall say that *almost all* functions from a class $F(n) \subseteq P_k(n)$ have a property R if the fraction of functions from $F(n)$ with property R tends to 1 as $n \to \infty$.

It is easy to prove the following assertion.

Lemma 2.1. Almost all functions $f \in P_k(n)$ are regular.

Definition 2.3. The set of vectors $T^*(s, f) \subseteq E_k^n$ (respectively, $T^{**}(s, f) \subseteq E_k^n$) is called an *s− test of sensitivity (stability)* for $f \in P_k(n)$, if for each subset $\{i_1, ..., i_r\} \subseteq N_n$, $1 \leq r \leq s$, the sensitivity (stability) of f with respect to $\{x_{i_1}, ..., x_{i_r}\}$ implies the existence of a vector $\tilde{\alpha} \in T^*(s, f)$ (respectively, $\tilde{\alpha} \in T^{**}(s, f)$) at which f is sensitive (stable) with respect to $\{x_{i_1}, ..., x_{i_r}\}$.

Definition 2.4. The set of vectors $T(s, f) \subseteq E_k^n$ is called a *(dynamic) s− test of regularity* for the function $f \in P_k(n)$, if it is both an $s-$ test of sensitivity and an $s-$ test of stability for f.

For $s = 1$ (respectively $s = n$) the $s-$ tests will be called *single (complete)* tests. The test $T_0(s, f)$ is called a *minimal* $s-$ test for f, if $|T_0(s, f)| = t(s, f) = \min\{|T(s, f)|/T(s, f) \in Z(s, f)\}$, where $Z(s, f)$ is the set of all $s-$ tests of regularity for f, and $|A|$ denotes the cardinality of the set A.

The main objective of this paper is to find bounds on the complexity measure $t(s, f)$ of minimal $s-$ tests of regularity for logic functions $f \in P_k(n), k \geq 2, 1 \leq s \leq n$.

3 The Complexity of Single Tests

The set of functions $P_k(n)$ may be considered as a probability space with every element $f \in P_k(n)$ having the same probability $\exp_k(-k^n)$. Denote by H_n the Hamming code in the n-cube E_2^n. It is known (see [9]) that $|H_n| = exp_2(n - [\log_2(n+1)])$. The pair of vectors $\tilde{\alpha}, \tilde{\beta} \in H_n, \tilde{\alpha} \neq \tilde{\beta}$, will be called a regular pair for the Boolean function $f \in P_2(n)$ iff $\bigwedge_{i=1}^n (f(\tilde{\alpha}) \oplus f(\tilde{\alpha}^{\{i\}}) \oplus f(\tilde{\beta}) \oplus f(\tilde{\beta}^{\{i\}})) = 1$ where $\oplus$ is the modulo 2 sum. Obviously, if $\{\tilde{\alpha}, \tilde{\beta}\}$ is a regular pair for $f \in P_2(n)$ then $\{\tilde{\alpha}, \tilde{\beta}\}$ is a single test of regularity for f.

For every $\tilde{\alpha}, \tilde{\beta} \in H_n, \tilde{\alpha} \neq \tilde{\beta}, f \in P_2(n)$ define the following random variables

$$\xi_{\tilde{\alpha},\tilde{\beta}}(f) = \begin{cases} 1 & \text{if } \{\tilde{\alpha}, \tilde{\beta}\} \text{ is a regular pair for } f \\ 0 & \text{otherwise} \end{cases}$$

Obviously, $\xi_2(f) = \sum_{\tilde{\alpha},\tilde{\beta} \in H_n, \tilde{\alpha} \neq \tilde{\beta}} \xi_{\tilde{\alpha},\tilde{\beta}}(f)$ determines the number of regular pairs for $f \in P_2(n)$.

From Definition 2.4 with $k \geq 3, s = 1$, it follows that if for every $i, 1 \leq i \leq n$, the function $f \in P_k(n)$ is both sensitive and stable at vector $\tilde{\alpha} \in E_k^n$ with respect to variable x_i, then $\{\tilde{\alpha}\}$ is a minimal single test of regularity for f.

For every $\tilde{\alpha} \in E_k^n$ and $f \in P_k(n)$ define the following random variable

$$\xi_{\tilde{\alpha}}(f) = \begin{cases} 1 & \text{if } \{\tilde{\alpha}\} \text{ is a single test for } f \\ 0 & \text{otherwise} \end{cases}$$

Obviously, the random variable $\xi_k(f) = \sum_{\tilde{\alpha} \in E_k^n} \xi_{\tilde{\alpha}}(f)$ determines the number of single tests of regularity for $f \in P_k(n)$, $k \geq 3$. Now let us compute the expectations $M\xi_k(f)$ and dispersions $D\xi_k(f)$ for the random variables $\xi_k(f), k \geq 2$.

Lemma 3.1

$$M\xi_k(f) = \begin{cases} |H_n|(|H_n| - 1)2^{-n-1}, & \text{if } k = 2 \\ k^n(1 - ((k-1)/k)^{k-1} - k^{-k+1})^n & \text{if } k \geq 3 \end{cases}$$

Lemma 3.2 .

$$D\xi_k(f) = \begin{cases} (1 - 2^{-n})M\xi_2(f) & \text{if } k = 2 \\ M\xi_k(f) + (c_1(k)n^2 + c_2(k)n - 1)k^{-n}(M\xi_k(f))^2 & \text{if } k \geq 3 \end{cases}$$

where $c_1(k)$ and $c_2(k)$ depend only on k.

Lemma 3.3. For almost all functions $f \in P_k(n), k \geq 2, n \to \infty$,

$$\xi_k(f) \sim M\xi_k(f).$$

Proof is based on the second - moment method (see, e.g., [8]). From Lemmas 3.1 and 3.2 it follows that $M\xi_k(f) \to \infty$, and $D\xi_k(f) = o((M\xi_k(f))^2)$. Let $\phi(n) \to \infty, \phi(n) = o(\sqrt{M\xi_k(f)})$, then according to Chebyshev's inequality (see [8]) the fraction of functions $f \in P_k(n)$ satisfying the inequality

$|\xi_k(f) - M\xi_k(f)| < (\phi(n))^{-1} M\xi_k(f)$ tends to 1 as $n \to \infty$. Consequently, by definition, $\xi_k(f) \sim M\xi_k(f)$ for almost all functions $f \in P_k(n)$, $n \to \infty$, $k \geq 2$.

Theorem 3.1 . For almost all functions $f \in P_k(n)$

$$t(1,f) = \begin{cases} 2 & \text{if } k = 2 \\ 1 & \text{if } k \geq 3 \end{cases}$$

Proof for $k \geq 3$ follows immediately from Lemma 3.3. For $k = 2$ from Lemma 3.3 it follows that $t(1,f) \leq 2$ for almost all functions $f \in P_2(n)$. From Lemma 2.1 and Definition 2.4 with $k = 2, s = 1$, it follows that $t(1,f) \geq 2$ which completes the proof.

4 Upper Bounds on the Complexity of $s-$ Tests

The set of vectors $Q \subseteq E_k^n$ will be called an $(n, 2s+1, r)-$ code if $|Q| = r$ and $\rho(\tilde{\alpha}, \tilde{\beta}) \geq 2s+1$ for all $\tilde{\alpha}, \tilde{\beta} \in Q, \tilde{\alpha} \neq \tilde{\beta}$, where $\rho(\tilde{\alpha}, \tilde{\beta})$, called the distance between $\tilde{\alpha}$ and $\tilde{\beta}$, is the number of coordinates $i, 1 \leq i \leq n$, such that $\alpha_i \neq \beta_i$.

Lemma 4.1 . Let $Q \subseteq E_k^n$ be an $(n, 2s+1, r)-$ code, $s \geq 2, k \geq 2$ and $\psi(n) \to \infty$ as $n \to \infty$. If

$$r = \begin{cases} \lfloor \log_2 \sum_{i=1}^{s} \binom{n}{i} + \psi(n) \rfloor & \text{for } k = 2 \\ \lfloor (k-1)^{-1} \log_{k/(k-1)} n + \psi(n) \rfloor & \text{for } k \geq 3 \end{cases}$$

then for almost all functions $f \in P_k(n)$ Q is an $s-$ test of regularity.

Proof. Let $I = \{i_1, ..., i_m\} \subseteq N_n, 1 \leq m \leq s$. Denote by $\Phi_k(I)$ (respectively, $\Psi_k(I)$) the number of functions $f \in P_k(n)$ that are not sensitive (stable) at each vector $\tilde{\alpha} \in Q$ with respect to the subset of variables $\{x_{i_1}, ..., x_{i_m}\}$. Then it is easy to compute

$$\Phi_k(I) = \exp_k(k^n - r(k-1)^m), \Psi_k(I) = k^{k^n} \exp_{(k-1)/k}(r(k-1)^m).$$

Let $p_k(n, Q)$ be the probability of an event that Q is an $s-$ test of regularity for a random function from $P_k(n)$. Now it is easy to verify that if r satisfies the conditions of the lemma, then

$$p_k(n, Q) \geq 1 - \exp_k(-k^n) \sum_{I \subseteq N_n, 1 \leq |I| \leq s} (\Phi_k(I) + \Psi_k(I)) =$$

$$= 1 - \sum_{i=1}^{s} \binom{n}{i} (\exp_k(-r(k-1)^i + \exp_{(k-1)/k}(r(k-1)^i)) = 1 - o(1).$$

Denote by $F_k(n, 2s+1)$ a code in E_k^n of maximal cardinality with a code distance $2s+1$. The following statement is a straight-forward generalization of a well-known result for $k = 2$ (see [9]).

Lemma 4.2 . For all $k \geq 2$

$$|F_k(n, 2s+1)| \geq k^n / \sum_{i=0}^{2s} \binom{n}{i} (k-1)^i.$$

Theorem 4.1 . For almost all functions $f \in P_2(n)$ and $n \to \infty$

$$t(s,f) \lesssim \begin{cases} s \log_2 \frac{n}{s} & \text{if } s = o(n) \\ nH(\lambda) & \text{if } s = \lfloor \lambda n \rfloor, 0 < \lambda < 1/4 \end{cases}$$

where $H(\lambda) = -\lambda \log_2 \lambda - (1-\lambda)\log_2(1-\lambda)$.

Proof. Applying the well - known (*see* [10]) inequality $\sum_{i=0}^{m} \binom{n}{i} \leq \exp_2(nH(m/n))$ where $m \leq n/2$, and taking into account Lemma 4.2 we obtain that if $s = \lfloor \lambda n \rfloor, 0 < \lambda < 1/4, n \to \infty$ and r satisfies the conditions of Lemma 4.1 for $k = 2$, then

$$|F_2(n, 2s+1)| \geq \exp_2(n(1 - H(2s/n))) > \exp_2(n(1 - H(2\lambda))) > r.$$

Thus, if the conditions mentioned above are satisfied, then there can be constructed an $(n, 2s+1, r)-$ code which according to Lemma 4.1 will be an s-test of regularity for almost all functions $f \in P_2(n)$. Hence, $t(s,f) \leq r = \lfloor \log_2 \sum_{i=1}^{s} \binom{n}{i} + \psi(n) \rfloor$, whence the proof follows directly.

Theorem 4.2 . For almost all functions $f \in P_k(n), k \geq 3$,

$$2 \leq s \leq \lfloor n(\log_2 k - 1)/(2\log_2(k-1)) \rfloor, n \to \infty,$$

$$t(s,f) \lesssim (k-1)^{-1} \log_{k/(k-1)} n.$$

Proof is analogous to that of Theorem 4.1.

5 Lower Bounds on the Complexity of $s-$ Tests

Let $\{i_1, ..., i_r\} \subseteq N_n, c_j \in E_k, j = \overline{1,r}, 1 \leq r \leq n, k \geq 2$. Denote by $E_k^n(i_1, c_1; ...; i_r, c_r)$ the set of all vectors $\tilde{\beta} \in E_k^n$ with $\beta_{i_j} = c_j$, $j = \overline{1,r}$, called an $(n-r)-$ dimensional subcube in E_k^n. The set of indices $\{i_1, ..., i_r\}$ will be called the set of *fixed indices* of the subcube. Any two subcubes in E_k^n will be called *parallel* if they have the same set of fixed indices. Obviously, any two parallel subcubes do not intersect, and $|E_k^n(i_1, c_1; ...; i_r, c_r)| = k^{n-r}$.

The following statement is a straightforward generalisation of a lemma from [10].

Lemma 5.1 . For any set $M \subseteq E_k^n, |M| = m \leq n+1, k \geq 2$, there exists a family of $(n-m+1)-$ dimensional parallel subcubes of cardinality k^{m-1}, with each subcube containing at most one vector from M.

Let $\pi_k(m)$ be the probability of an event that a random function from $P_k(n)$ has an $s-$ test of regularity consisting of m vectors.

Lemma 5.2 . For $k \geq 2, m \leq n+1$ and $n \to \infty$

$$\pi_k(m) \leq \binom{k^n}{m} \prod_{i=1}^{s} \left(1 - (\frac{k-1}{k})^{m(k-1)^i} - k^{-m(k-1)^i}\right)^{\binom{n-m+1}{i}} + o(1).$$

Proof. Denote by $\mathcal{M}(M)$ the set of functions $f \in R_k(n)$ having M as an $s-$ test of regularity. Then, taking into account Lemma 2.1 , it is easy to verify that

$$\pi_k(m) \leq \exp_k(-k^n) \sum_{M \subseteq E_k^n, |M|=m} |\mathcal{M}(M)| + o(1).$$

According to Lemma 5.1 there can be found a family of $(n-m+1)-$ dimensional parallel subcubes of cardinality k^{m-1} with each subcube containing at most one vector from M. Let $\{j_1, j_2, ..., j_{m-1}\}$ be the set of fixed indices corresponding to every subcube from the family. Denote $N_n^* = N_n \setminus \{j_1, ..., j_{m-1}\}$. Let $\mathcal{M}^*(M)$ be the set of functions $f \in R_k(n)$ having M as an $s-$ test of regularity with respect to N_n^*, i.e., for every subset $\{l_1, ..., l_v\} \subseteq N_n^*$, $1 \leq v \leq s$, f is both sensitive and stable with respect to the set of variables $\{x_{l_1}, ..., x_{l_v}\}$. Obviously, $\mathcal{M}(M) \subseteq \mathcal{M}^*(M)$.

It is easy to compute that

$$|\mathcal{M}^*(M)| \leq \prod_{i=1}^{s} (\exp_k(m(k-1)^i) - \exp_{k-1}(m(k-1)^i) - 1)^{\binom{n-m+1}{i}} \times$$

$$\times \exp_k\left(k^n - m \sum_{i=1}^{s} \binom{n-m+1}{i} (k-1)^i\right) =$$

$$k^{k^n} \prod_{i=1}^{s} \left(1 - \left(\frac{k-1}{k}\right)^{m(k-1)^i} - k^{-m(k-1)^i}\right)^{\binom{n-m+1}{i}}.$$

Whence the proof of the lemma follows immediately.

The proof of the following statement is obvious.

Lemma 5.3 . If $\pi_k(m) = o(1)$ for $k \geq 2, m = m(n), n \to \infty$, then for almost all functions $f \in P_k(n)$

$$t(s, f) \geq m + 1.$$

Theorem 5.1 . For $n \to \infty$ and almost all functions $f \in P_2(n)$

$$t(s, f) \gtrsim \begin{cases} (s-1)\log_2 n & \text{if } s = const \geq 2 \\ s \log_2 \frac{n}{s} & \text{if } s = o(n), s \to \infty \\ nH(\lambda)/(1+H(\lambda)) & \text{if } s = \lfloor \lambda n \rfloor, 0 < \lambda < 1/2 \\ n/2 & \text{if } s \geq n/2 \end{cases}$$

Proof. From Lemma 5.2 with $k = 2$ we obtain

$$\pi_k(m) \leq 2^{mn} \exp_e\left(-2^{-m+1} \sum_{i=1}^{s} \binom{n-m+1}{i}\right) + o(1).$$

Putting

$$m_0 = \begin{cases} \lfloor (s-1)\log_2 n - \log_2 \log_2 n - \tau(n) \rfloor, & \\ \tau(n) = o(\log n), \tau(n) \to \infty & \text{if } s = const \geq 2 \\ \lfloor s \log_2 \frac{n}{s} - 2\log_2 n + s \log_2 (1 - \frac{s}{n} \log_2 \frac{n}{s}) \rfloor & \text{if } s = o(n), s \to \infty \\ \lfloor nH(\lambda)/(1+H(\lambda)) - 3\log_2 n \rfloor & \text{if } s = \lfloor \lambda n \rfloor, 0 < \lambda < 1/2 \\ \lfloor n/2 - \nu \log_2 n \rfloor, \nu = const > 5/4 & \text{if } s \geq n/2 \end{cases}$$

it is easy to verify that $\pi_k(m_0) = o(1), n \to \infty$. Hence in view of Lemma 5.3 we obtain the assertion of the theorem.

Corollary 5.1 . If $s = const \geq 2, n \to \infty$, then for almost all functions $f \in P_2(n)$,

$$t(s, f) \asymp \log n.$$

Corollary 5.2 . If $s = o(n), s \to \infty, n \to \infty$, then for almost all functions $f \in P_2(n)$,

$$t(s, f) \sim s \log_2 \frac{n}{s}.$$

Corollary 5.3 . If $s = \lfloor \lambda n \rfloor, 0 < \lambda < 1/4, n \to \infty$, then for almost all functions $f \in P_2(n)$,

$$t(s, f) \asymp n.$$

Corollaries 5.1-5.3 are obtained from Theorems 4.1 and 5.1.

Theorem 5.2 . If $s \geq 2, k \geq 3, n \to \infty$, then for almost all functions $f \in P_k(n)$,

$$t(s, f) \gtrsim (k-1)^{-2} \log_{k/(k-1)} n.$$

Proof. From Lemma 5.2 with $k \geq 3$ we obtain

$$\pi_k(m) \leq k^{mn} \exp_e \left(- \binom{n-m+1}{2} \left(\frac{k-1}{k}\right)^{m(k-1)^2} \right) + o(1).$$

It is easy to verify that $\pi_k(m) = o(1)$ for $m = \lfloor (k-1)^{-2} \log_{k/(k-1)} n - \log_k \log_k n \rfloor$, $n \to \infty$. Thus, in view of Lemma 5.3, the theorem is proved.

Corollary 5.4 . If $k \geq 3, 2 \leq s \leq \lfloor n(\log_2 k - 1)/(2 \log_2 (k-1)) \rfloor, n \to \infty$, then for almost all functions $f \in P_k(n)$,

$$t(s, f) \asymp \log n.$$

Proof follows directly from Theorems 4.2 and 5.2 .

6 Upper Bounds on the Complexity of Complete Tests

For Boolean functions $f \in P_2(n)$ denote $\omega_I^f(\tilde{x}) = f(\tilde{x}) \oplus f(\tilde{x}^I), I \subseteq N_n, I \neq \emptyset$. Now let us describe an algorithm for constructing a complete test of regularity for an arbitrary function $f \in P_2(n)$.

Algorithm 6.1 . **Step 1.** Choose an arbitrary vector $\tilde{\alpha}_1 \in E_2^n$ and put

$$T_1(f) = \{\tilde{\alpha}_1\};$$

$$T_1^0 = \{I / I \subseteq N_n, I \neq \emptyset, \omega_I^f(\tilde{\alpha}_1) = 0\};$$

$$T_1^1 = \{I/I \subseteq N_n, I \neq \emptyset, \omega_I^f(\tilde{\alpha}_1) = 1\}.$$

If $T_1^0 \bigcup T_1^1 \neq \emptyset$ and there exists a subset $I_2 \in T_1^\sigma$ for some $\sigma \in \{0,1\}$ such that $\omega_{I_2}^f(\tilde{x}) \not\equiv \sigma$, then we pass to the next step, otherwise the algorithm terminates.

Step $i (i \geq 2)$. Choose a vector $\tilde{\alpha}_i \in E_2^n$ such that $\omega_{I_i}^f(\tilde{\alpha}_i) = \bar{\sigma}$, where $I_i \in T_{i-1}^\sigma, \omega_{I_i}^f(\tilde{x}) \not\equiv \sigma, \sigma \in \{0,1\}$. If

$$\sum_{\sigma=0}^{1} |\{I/I \in T_{i-1}^\sigma, \omega_I^f(\tilde{\alpha}_i) = \bar{\sigma}\}| \geq \lfloor \frac{1}{2} \sum_{\sigma=0}^{1} |T_{i-1}^\sigma| \rfloor + 1$$

then put

$$T_i(f) = T_{i-1}(f) \bigcup \{\tilde{\alpha}_i\};$$

$$T_i^0 = \{I/I \in T_{i-1}^0, \omega_I^f(\tilde{\alpha}_i) = 0\};$$

$$T_i^1 = \{I/I \in T_{i-1}^1, \omega_I^f(\tilde{\alpha}_i) = 1\},$$

otherwise

$$T_i(f) = \{\tilde{\alpha}^{I_i}/\tilde{\alpha} \in T_{i-1}(f)\} \bigcup \{\tilde{\alpha}_i^{I_i}\};$$

$$T_i^0 = \{I \Delta I_i/I \in T_{i-1}^\sigma, I \neq I_i, \omega_I^f(\tilde{\alpha}_i) = \bar{\sigma}\};$$

$$T_i^1 = \{I \Delta I_i/I \in T_{i-1}^{1-\sigma}, \omega_I^f(\tilde{\alpha}_i) = \sigma\},$$

where Δ is the set-theoretical operation of symmetric difference.

If $T_i^0 \bigcup T_i^1 \neq \emptyset$ and there exists a subset $I_{i+1} \in T_i^\sigma$ for some $\sigma \in \{0,1\}$ such that $\omega_{I_{i+1}}^f(\tilde{x}) \not\equiv \sigma$, then we pass to Step $i+1$, otherwise the algorithm terminates.

Finally, Algorithm 6.1 will determine a set $T_m(f)$ of $m \geq 1$ vectors which, as we are going to prove below, is a complete test of regularity for $f \in P_2(n)$.

We shall say that the subset $I \subseteq N_n, I \neq \emptyset$, is a feasible fault of sensitivity (stability) for $f \in P_2(n)$ if $\omega_I^f(\tilde{x}) \not\equiv 0$ (respectively, $\omega_I^f(\tilde{x}) \not\equiv 1$), and the vector $\tilde{\alpha} \in E_2^n$ detects the fault of sensitivity (stability) I for f if $\omega_I^f(\tilde{\alpha}) = 1$ (respectively, $\omega_I^f(\tilde{\alpha}) = 0$).

Theorem 6.1 . For all functions $f \in P_2(n)$,

$$t(n,f) \leq n+1.$$

Proof. Let $m = m(f)$ be the number of steps performed by Algorithm 6.1 for f. It is easy to see that for each $i, 1 \leq i \leq m$, the vectors from $T_i(f)$ do not detect the faults of sensitivity $I \in T_i^0$ and the faults of stability $J \in T_i^1$, and the total number of faults not detected by the vectors from $T_i(f)$ is reduced more than twice after each step. Since the algorithm terminates iff $T_m(f)$ detects all feasible faults of sensitivity and stability for f, then $T_m(f)$ is a complete test of regularity for f. Consequently, $t(n,f) \leq |T_m(f)| = m$. It is easy to prove by induction on $i, 1 \leq i \leq m$, that $|T_i^0| + |T_i^1| \leq 2^{n+1-i} - 1$. The conditions causing Algorithm 6.1 to terminate imply $0 \leq |T_m^0| + |T_m^1| \leq 2^{n+1-m} - 1$ whence the bound $m \leq n+1$ is derived directly.

Corollary 6.1 . For almost all functions $f \in P_2(n)$,

$$n/2 \lesssim t(n, f) \leq n + 1.$$

Proof follows directly from Theorems 5.1 and 6.1.

Let $t^*(s, f)$ (respectively, $t^{**}(s, f)$) be the complexity of a minimal $s-$ test of sensitivity (stability) for $f \in P_k(n)$.

Lemma 6.1 [7]. For almost all functions $f \in P_k(n), k \geq 3$,

$$t^*(n, f) \leq n.$$

We will say that the function $f \in P_k(n), k \geq 3$, is stable in E_2^n if for every $I \subseteq N_n, I \neq \emptyset$, there exists a vector $\tilde{\alpha} \in E_2^n$ detecting the fault of stability I for f in E_2^n, i.e. , $f(\tilde{\alpha}) = f(\tilde{\alpha}^I)$, where $\tilde{\alpha}^I$ is the sole vector from $G_I(\tilde{\alpha}) \bigcap E_2^n$. Denote by $\theta(f, I)$ the number of vectors $\tilde{\alpha} \in E_2^n$ detecting the fault of stability I for f.

Lemma 6.2 . For almost all functions $f \in P_k(n), k \geq 3$, the inequality $\theta(f, I) \geq \frac{1}{k} 2^{n-1}$ holds for all $I \subseteq N_n, I \neq \emptyset$.

Lemma 6.3 . Almost all functions $f \in P_k(n), k \geq 3$, are stable in E_2^n.

Lemma 6.4 . Almost all functions $f \in P_k(n), k \geq 3$, take all k values from E_k at vectors from E_2^n.

Proofs of Lemmas 6.2-6.4 are not difficult, so they are omitted.

Now let us describe an algorithm for constructing a complete test of stability for almost all functions $f \in P_k(n), k \geq 3$. To this end, each function $f \in P_k(n)$ is associated to a table $J(f)$ with 2^n rows, one for each vector from E_2^n, and $2^n - 1$ columns, one for each feasible fault of stability $I \subseteq N_n, I \neq \emptyset$. At the intersection of the ith row and jth column corresponding to $\tilde{\alpha} \in E_2^n$ and $I \subseteq N_n$, respectively, there stands a '1'('0') iff $f(\tilde{\alpha}) = f(\tilde{\alpha}^I)$ (respectively, $f(\tilde{\alpha}) \neq f(\tilde{\alpha}^I)$). Let $T_0(f) = \emptyset$ and $J_0(f) = J(f)$.

Algorithm 6.2 . **Step i** ($i \geq 1$) . Select a vector $\tilde{\alpha}_i \in E_2^n$ with the corresponding row in $J_{i-1}(f)$ having the maximum number of 1's, and put $T_i(f) = T_{i-1}(f) \bigcup \{\tilde{\alpha}_i\}$. Denote by $J_i(f)$ the table obtained from $J_{i-1}(f)$ by deleting all the columns having 1's in the row corresponding to $\tilde{\alpha}_i$. If $J_i(f) = \emptyset$ or $J_i(f)$ has only 0's , then the algorithm terminates, otherwise we pass to Step $i + 1$.

Note that according to Lemma 6.3, for almost all functions $f \in P_k(n)$ Algorithm 6.2 terminates iff $J_m(f) = \emptyset$ for some $m \geq 1$. Hence, the following assertion holds.

Lemma 6.5 . For almost all functions $f \in P_k(n), k \geq 3$, Algorithm 6.2 constructs a complete test of stability.

Lemma 6.6 . For almost all functions $f \in P_k(n), k \geq 3$,

$$t^{**}(n, f) \leq \lceil n \log_{2k/(2k-1)} 2 \rceil + 1.$$

Proof. Let μ_r be the fraction of faults of stability $I \subseteq N_n$, $I \neq \emptyset$, detected by vectors $\tilde{\alpha}_1, \ldots, \tilde{\alpha}_r \in E_2^n$ which are selected after the rth step of Algorithm 6.2. Then, obviously, $(1 - \mu_r)(2^n - 1)$ is the number of faults of stability remaining still undetected , i.e., the number of nonempty subsets $I \subseteq N_n$ such that $f(\tilde{\alpha}_j) \neq f(\tilde{\alpha}_j^I)$ for all $j, 1 \leq j \leq r$. From Lemmas 6.2 and 6.3 it follows that for

almost all functions $f \in P_k(n)$ the total number of feasible faults of stability, detectable by the remaining vectors from E_2^n is not less than $\frac{1}{k}(1-\mu_r)(2^n-1)2^{n-1}$. Consequently, among the remaining vectors there can be found a vector detecting not less than $(1-\mu_r)(2^n-1)/(2k)$ faults of stability which are not detected by the first r selected vectors. Thus, we obtain

$$\mu_{r+1} \geq \mu_r + \frac{1}{2k}(1-\mu_r) = (1-\frac{1}{2k})\mu_r + \frac{1}{2k} \geq \ldots \geq$$

$$\geq (1-\frac{1}{2k})^r \mu_1 + \frac{1}{2k}\sum_{i=0}^{r-1}(1-\frac{1}{2k})^i \geq$$

(since Lemma 6.4 implies $\mu_1 \geq \frac{1}{k} > \frac{1}{2k}$)

$$\geq \frac{1}{2k}\sum_{i=0}^{r}(1-\frac{1}{2k})^i = 1-(1-\frac{1}{2k})^{r+1}.$$

Thus, $\mu_r \geq 1-(1-\frac{1}{2k})^r$. Putting $r_0 = \lceil \log_{1-\frac{1}{2k}} \frac{1}{2^n-1} \rceil \leq \lceil n \log_{2k/(2k-1)} 2 \rceil$, we find out that after the choice of r_0 vectors there will still remain undetected not more than

$$(1-\mu_{r_0})(2^n-1) \leq (1-\frac{1}{2k})^{r_0}(2^n-1) \leq 1$$

faults of stability. Taking into account Lemma 6.5, we obtain that for almost all functions $f \in P_k(n)$

$$t^{**}(n,f) \leq r_0 + (1-\mu_{r_0})(2^n-1) \leq \lceil n \log_{2k/(2k-1)} 2 \rceil + 1.$$

Theorem 6.2 . For almost all functions $f \in P_k(n), k \geq 3$,

$$t(n,f) \stackrel{\sim}{\leq} n(1+\log_{2k/(2k-1)} 2).$$

Proof follows directly from Lemmas 6.1 and 6.6 .

Acknowledgment

The author would like to thank Z.Ésik (Hungary), Chairman of the Program Committee of the 9th International Conference Fundamentals of Computation Theory (FCT'93) for his support.

The author also thanks T.Gaiser (Hungary) of the Organising Committee of FCT'93 and R.Maroutian of the Institute of Informatics and Automation Problems, Armenian National Academy of Sciences for their technical assistance.

References

[1] Si S.-C. Dynamic testing of redundant logic networks . IEEE Trans. on Computers, v. C-27, N.9 (1978), pp. 828-832.

[2] Yuan Y., Chen T. The dynamic testing of combinational logic networks. 12th Annu. Int. Symp. Fault-Tolerant Comput. (Santa-Monica, Calif., June 22-24, 1982) New York : s.l.(1982), pp. 173-180.

[3] Petrossian A.V. Some differential properties of Boolean functions. Tanulmányok - MTA Számitástech. Automat. Kutató Int., v. 135 (1982), pp. 15-37 (Russian)

[4] Vardanian V.A. On the complexity of dynamic tests for Boolean functions. Doklady Akademii Nauk Armenii, Yerevan, v. 77, N.3, (1983), pp. 113-116 (Russian).

[5] Vardanian V.A. On the length of single dynamic tests for monotone Boolean functions. In: Proc. 5th Int. Conf. FCT'85, Lecture Notes in Computer Science, v. 199, (1985), pp. 442 - 449.

[6] Vardanian V.A. On the complexity of activity tests for partial Boolean functions. Avtomatika i telemechanika, Moskva : Nauka, N.7 (1986), pp. 118-124 (Russian).

[7] Vardanian V.A. On the complexity of dynamic tests for k-valued logic functions. Kibernetika, Kiev, N.3 (1988), pp.29-36 (Russian).

[8] Erdõs P., Spencer J. Probabilistic methods in combinatorics. Moskva : Mir, 1976 (Russian).

[9] Levenshtein V.I. The elements of the coding theory. In: Diskretnaya matematika i matematicheskiye voprosy kibernetiki, Moskva : Nauka, v. 1 (1974), pp. 207-305 (Russian).

[10] Noskov V.N. On the complexity of tests controlling the work of inputs of logic circuits. Diskretniy analiz, Novosibirsk, v.25 (1975). pp. 23-51 (Russian).

Received March 5, 1994

Acta Cybernetica, Vol. 11, No. 4, Szeged, 1994

Some Remarks on Functional Dependencies in Relational Datamodels*

Vu Duc Thi † Le Thi Thanh †

Abstract

The concept of minimal family is introduced. We prove that this family and family of functional dependencies (*FDs*) determine each other uniquely. A characterization of this family is presented.

We show that there is no polynomial time algorithm finding a minimal family from a given relation scheme. We prove that the time complexity of finding a minimal family from a given relation is exponential in the number of attributes.

Key Words and Phrases: relation, relational datamodel, functional dependency, relation scheme, closure, closed set, minimal generator, key, minimal key, antikey.

1 Introduction

The functional dependency introduced by E.F.Codd is one of important semantic constraints in the relational datamodel.

The family of FDs has been widely studied in the literature. In this paper we give a family of sets and show that it is determined uniquelyby family of FDs. This paper presents some results about computational problems related to this family.

Let us give some necessary definitions and results used in what follows.

Let $R = \{a_1, \ldots, a_n\}$ be a nonempty finite set of attributes. A functional dependency is a statement of the form $A \to B$, where $A, B \subseteq R$. The FD $A \to B$ holds in a relation $r = \{h_1, \ldots, h_m\}$ over R if $\forall h_i, h_j \in r$ we have $h_i(a) = h_j(a)$ for all $a \in A$ implies $h_i(b) = h_j(b)$ for all $b \in B$. We also say that r satisfies the FD $A \to B$.

Let F_r be a family of all FDs that hold in r. Then $F = F_r$ satisfies

(1) $A \to A \in F$,

(2) $(A \to B \in F, B \to C \in F) \Longrightarrow (A \to C \in F)$,

(3) $(A \to B \in F, A \subseteq C, D \subseteq B) \Longrightarrow (C \to D \in F)$,

*Research supported by Hungarian Foundation for Scientific Research Grant 2575.

†Computer and Automation Institute Hungarian Academy of Sciences, H-1111 Budapest, Lágymányosi u. 11. Hungary.

(4) $(A \to B \in F, C \to D \in F) \Longrightarrow (A \cup C \to B \cup D \in F)$.

A family of FDs satisfying (1)-(4) is called an f-family (sometimes it is called the full family) over R.

Clearly, F_r is an f-family over R. It is known [1] that if F is an arbitrary f-family, then there is a relation r over R such that $F_r = F$.

Given a family F of FDs, there exists a unique minimal f-family F^+ that contains F. It can be seen that F^+ contains all FDs which can be derived from F by the rules (1)-(4).

A relation scheme s is a pair $< R, F >$, where R is a set of attributes, and F is a set of FDs over R. Denote $A^+ = \{a: A \to \{a\} \in F^+\}$. A^+ is called the closure of A over s. It is clear that $A \to B \in F^+$ iff $B \subseteq A^+$.

Clearly, if $s =< R, F >$ is a relation scheme, then there is a relation r over R such that $F_r = F^+$ (see, [1]). Such a relation is called an Armstrong relation of s.

Let R be a nonempty finite set of attributes and $P(R)$ its power set. The mapping $H : P(R) \to P(R)$ is called a closure operation over R if for all $A, B \in P(R)$, the following conditions are satisfied :

(1) $A \subseteq H(A)$,

(2) $A \subseteq B$ implies $H(A) \subseteq H(B)$,

(3) $H(H(A)) = H(A)$.

Let $s =< R, F >$ be a relation scheme. Set $H_s(A) = \{a : A \to \{a\} \in F^+\}$, we can see that H_s is a closure operation over R.

Let r be a relation, $s =< R, F >$ be a relation scheme. Then A is a key of r (a key of s) if $A \to R \in F_r$ ($A \to R \in F^+$). A is a minimal key of $r(s)$ if A is a key of $r(s)$ and any proper subset of A is not a key of $r(s)$.

Denote $K_r(K_s)$ the set of all minimal keys of $r(s)$.

Clearly, K_r, K_s are Sperner systems over R, i.e. $A, B \in K_r(K_s)$ implies $A \not\subseteq B$.

Let K be a Sperner system over R. We define the set of antikeys of K, denoted by K^{-1}, as follows:

$$K^{-1} = \{A \subset R : (B \in K) \Longrightarrow (B \not\subseteq A) \text{ and } (A \subset C) \Longrightarrow (\exists B \in K)(B \subseteq C)\}.$$

It is easy to see that K^{-1} is also a Sperner system over R.

It is known [5] that if K is an arbitrary Sperner system over R, then there is a relation scheme s such that $K_s = K$.

In this paper we always assume that if a Sperner system plays the role of the set of minimal keys (antikeys), then this Sperner system is not empty (doesn't contain R). We consider the comparison of two attributes as an elementary step of algorithms. Thus, if we assume that subsets of R are represented as sorted lists of attributes, then a Boolean operation on two subsets of R requires at most $|R|$ elementary steps.

Let $L \subseteq P(R)$. L is called a meet-irreducible family over R (sometimes it is called a family of members which are not intersections of two other members) if $\forall A, B, C \in L$, then $A = B \cap C$ implies $A = B$ or $A = C$.

Let $I \subseteq P(R)$, $R \in I$, and $A, B \in I \Longrightarrow A \cap B \in I$. I is called a meet-semilattice over R. Let $M \subseteq P(R)$. Denote $M^+ = \{\cap M' : M' \subseteq M\}$. We say that M is a generator of I if $M^+ = I$. Note that $R \in M^+$ but not in M, by convention it is the intersection of the empty collection of sets.

Denote $N = \{A \in I : A \neq \cap\{A' \in I : A \subset A'\}\}$.

In [5] it is proved that N is the unique minimal generator of I.

It can be seen that N is a family of members which are not intersections of two other members.

Let H be a closure operation over R. Denote $Z(H) = \{A : H(A) = A\}$ and $N(H) = \{A \in Z(H) : A \neq \cap\{A' \in Z(H) : A \subset A'\}\}$. $Z(H)$ is called the family of closed sets of H. We say that $N(H)$ is the minimal generator of H.

It is shown [5] that if L is a meet-irreducible family then L is the minimal generator of some closure operation over R. It is known [1] that there is an one-to-one correspondence between these families and f-families.

Let r be a relation over R. Denote $E_r = \{E_{ij} : 1 \leq i < j \leq |r|\}$, where $E_{ij} = \{a \in R : h_i(a) = h_j(a)\}$. Then E_r is called the equality set of r.

Let $T_r = \{A \in P(R) : \exists E_{ij} = A, \nexists E_{pq} : A \subset E_{pq}\}$. We say that T_r is the maximal equality system of r.

Let r be a relation and K a Sperner system over R. We say that r represents K if $K_r = K$.

The following theorem is known ([7])

Theorem 1.1 *Let K be a non-empty Sperner system and r a relation over R. Then r represents K iff $K^{-1} = T_r$, where T_r is the maximal equality system of r.*

In [6] we proved the following theorem.

Theorem 1.2 *Let $r = \{h_1, \ldots, h_m\}$ be a relation, and F an f-family over R. Then $F_r = F$ iff for every $A \subseteq R$*

$$H_F(A) = \begin{cases} \bigcap\limits_{A \subseteq E_{ij},} E_{ij} & \text{if } \exists E_{ij} \in E_r : A \subseteq E_{ij} \\ R & \text{otherwise,} \end{cases}$$

where $H_F(A) = \{a \in R : A \to \{a\} \in F\}$ and E_r is the equality set of r.

2 Results

In this section we introduce the concept of minimal family. We show that this family and family of FDs determine each other uniquely. We give some desirable properties of this family. We present some results about the relationship between this family, meet-semiattice and family of FDs.

Definition 2.1 *Let $Y \subseteq P(R) \times P(R)$. We say that Y is a minimal family over R if the following conditions are satisfied :*

(1) $\forall (A, B), (A', B') \in Y : A \subset B \subseteq R$, $A \subset A'$ *implies* $B \subset B'$, $A \subset B'$ *implies* $B \subseteq B'$.

(2) Put $R(Y) = \{B : (A, B) \in Y\}$. *For each* $B \in R(Y)$ *and* C *such that* $C \subset B$ *and* $\nexists B' \in R(Y) : C \subset B' \subset B$, *there is an* $A \in L(B) : A \subseteq C$, *where* $L(B) = \{A : (A, B) \in Y\}$.

Remark 2.2 *(1.)* $R \in R(Y)$.

(2.) From $A \subset B'$ *implies* $B \subseteq B'$ *there is no a* $B' \in R(Y)$ *such that* $A \subset B' \subset B$ *and* $A = A'$ *implies* $B = B'$.

(3.) Because $A \subset A'$ *implies* $B \subset B'$ *and* $A = A'$ *implies* $B = B'$, *we can be see that* $L(B)$ *is a Sperner system over* R *and by (2)* $L(B) \neq \emptyset$.

Let I be a meet-semilattice over R.

Put $M^*(I) = \{(A, B) : \exists C \in I : A \subset C,\ A \neq \cap\{C : C \in I,\ A \subset C\}, B = \cap\{C : C \in I,\ A \subset C\}\}$.

Set $M(I) = \{(A, B) \in M^*(I) : \nexists (A', B) \in M^*(I) : A' \subset A\}$.

Note that if $C \in I$, then C is an one-term intersection. It is possible that $A = \emptyset$.

It can be seen that for any meet-semilattice I there is exactly one family $M(I)$.

Theorem 2.3 *Let* I *be a meet-semilattice over* R. *Then* $M(I)$ *is a minimal family over* R.

Conversely, if Y *is a minimal family over* R, *then there is exactly one meet-semilattice* I *so that* $M(I) = Y$, *where* $I = \{C \subseteq R : \forall (A, B) \in Y : A \subseteq C$ *implies* $B \subseteq C\}$.

Proof: Assume that I is a meet-semilattice over R. We have to show that $M(I)$ is a minimal family over R. It is obvious that $A \subset B \subseteq R$.

From $B' = \cap\{D : D \in I,\ A' \subset D)\}$, we have $B' \subseteq D$. If $A \subset B'$, then $A \subset D$ and by $B = \cap\{C : C \in I : A \subset C\}$ we obtain $B \subseteq B'$. By $\nexists (A', B) \in M^*(I) : A' \subset A$ and from $A' \subset A \subset B$ implies $B' \subseteq B$ we can see that if $A' \subset A$ then $B' \subset B$. Thus, we obtain (1). Clearly, $L_I(B) = \{A : (A, B) \in M(I)\}$ is a Sperner system over R.

If there is a $B \in R(M(I))$ and D satisfying $D \subset B$ and $\forall B' \in R(M(I)) : D \subset B'$, $B' \subseteq B$ imply $B = B'$, then for all $A \in L_I(B) : A \not\subseteq D(*)$.

It can be seen that $D \neq \cap\{C : C \in I, D \subset C\}$ and $B = \cap\{C : C \in I, D \subset C\}$.

If $L_I(B) \cup D$ is a Sperner system over R, then by definition of $M(I)$ we have $D \in L_I(B)$. From $(*)$ this is a contradiction.

If there exists an $A \in L_I(B) : D \subset A$, then this conflicts with the definition of $M(I)$. Thus, we have (2) in Definition 2.1. Consequently, $M(I)$ is a minimal family over R.

Conversely, Y is a minimal family over R. Clearly, I is a meet-semilattice over R. It is obvious that $(A, B) \in Y$ implies $A \notin I$.

Now we have to prove that $M(I) = Y$. Assume that $(A, B) \in Y$. By (1) in Definition 2.1 $\forall (A', B') \in Y : A' \subset B$ implies $B' \subseteq B$. From this and definition of I we obtain $B \in I$.

According to definition of I there is no $C \in I$ such that $A \subset C \subset B$. On the other hand, $A \subset B$ and B is an intersection of Cs, where $C \in I$, $A \subset C$. Thus, $B = \cap\{C : C \in I, A \subset C\}$ and $A \neq \cap\{C : C \in I, A \subset C\}$. Hence, $(A, B) \in M^*(I)$ holds.

Clearly, if $A = \emptyset$ then $(A, B) \in M(I)$. Assume that $A \neq \emptyset$ and $(A', B) \in M^*(I)$. It is obvious that by the definition of $M^*(I)$ $A' \subset B$ and $\nexists B' : A' \subset B' \subset B$. By (2) in Definition 2.1 there is an $A'' \in L(B) : A'' \subseteq A'$. Because $L(B)$ is a Sperner system over R and $A \in L(B)$ we have $A' \not\subseteq A$. Thus, $(A, B) \in M(N)$ holds.

Suppose that $A \subset R$ and $A \notin I$. Based on the above proof, $B \in R(Y)$ implies $B \in I$. Clearly, $R \in R(Y)$. Consequently, for A there is a $B \in R(Y)$ such that $A \subset B\,(**)$. We choose a set B so that $|B|$ is minimal for $(**)$, i.e. $\nexists B' \in R(Y) : A \subset B' \subset B$. According to (2) in Definition 2.1 there exists an $A' \in L(B) : A' \subseteq A$. If there is $C \in I : A \subset C \subset B$, then $A' \subset C \subset B$. This conflicts with the definition

of I. Consequently, for all $C \in I$ and $C \neq B$, $A \subset C$ implies $B \subset C$. From this and according to the definition of $M^*(I)$ $(A, B) \in M^*(I)$ implies $B \in R(Y)$.

Assume that $(A, B) \in M(I)$. By the above proof, $B \in R(Y)$ holds. We consider the set $L(B) = \{A' : (A', B) \in Y\}$. According to definition $M(I)$ we have $A \subset B$ and $\nexists B' \in R(Y) : A \subset B' \subset B$. By (2) in Definition 2.1 there is an $A' \in L(B)$ such that $A' \subseteq A$. If $A' \subset A$, then according to the above proof $(A', B) \in Y$ implies $(A', B) \in M(N)$. $A' \subset A$ contradicts the definition of $M(N)$. Thus, $A' = A$ holds. Consequently, we obtain $(A, B) \in Y$.

Suppose that there is a meet-semilattice I' such that $M(I') = Y$. We have to show that $I = I'$. By definition of $M(I')$ $E \in I'$ implies $E \in I$. Thus, $I' \subseteq I$ holds. Suppose that there is a $D \in I$ and $D \notin I'$. According to the definition of meet-semilattice $R \in I'$. Put $D'' = \cap\{E \in I' : D \subset E\}$. By $D \notin I'$ we have $D \subset D''$. According to $M^*(I')$ $(D, D'') \in M^*(I')$. From definition of $M(I')$ there is a $D' : D' \subseteq D$ and $(D', D'') \in M(I')$. Thus, $D' \subseteq D \subset D''$ holds. This conflicts with the fact that $D \in I$. Hence, $I = I'$ holds. □

It is known [1] that there is an one-to-one correspondence between families of FDs and meet-semilattices and by Theorem 2.3 we obtain the following.

Proposition 2.4 *There is an one-to-one correspondence between minimal families and families of FDs.*

Because there are one-to-one correspondences between meet-irredundant families, closure operations and families of FDs, we also have the following.

Proposition 2.5 *There are one-to-one correspondences between minimal families, meet-irredundant families and closure operations.*

Remark 2.6 *Let $s =< R, F >$ be a relation scheme over R. A functional dependency $A \rightarrow B \in F^+$ is called basic of s if*

(1) $A \subset B$,

(2) $\nexists A' : A' \subset A$ *and* $A' \rightarrow B \in F^+$,

(3) $\nexists B' : B \subset B'$ *and* $A \rightarrow B' \in F^+$,

Denote by $B(s)$ the set of all basic FDs of s.

If a relation scheme is changed to a relation we have a basic functional dependency of r. Denote the set of all basic FDs of r by $B(r)$.

It can be seen that the set $\{A \rightarrow R : A \in K_s\}$ is a subset of $B(s)$.

Remark 2.7 *Let $s =< R, F >$ be a relation scheme over R. Put $Z(s) = \{A : A^+ = A\}$. $Z(s)$ is a meet-semilattice over R. $M(Z(s))$ is called the minimal family of s. According to definitions of $M(I)$ and $B(s)$ we can see that $M(Z(s)) = \{(A, B) : A \rightarrow B \in B(s)\}$.*

It is known [17] that there is no a polynomial time algorithm finding a set of all minimal keys of a given relation scheme. From this and by Remark 2.6 we have the following corollary.

Corollary 2.8 *Let $s =< R, F >$ be a relation scheme over R. There is no a polynomial time algorithm to find the minimal family of s.*

Definition 2.9 *Let R be a relation over R and F_r a family of all FDs that hold in r. Put $A_r^+ = \{a : A \to \{a\} \in F_r\}$. Set $Z_r = \{A : A = A_r^+\}$. Then $M(Z_r)$ is called the minimal family of r.*

It is easy to see that the set $\{A \to R : A \in K_r\}$ is a subset of $B(r)$.

We construct a following exponential time algorithm finding a minimal family of a given relation.

In relation scheme $s =< R, F >$, a functional dependency $A \to B \in F$ is called redundant if either $A = B$ or there is $C \to B \in F$ such that $C \subseteq A$.

Algorithm 2.10 *(Finding a minimal family of r)*

(Input:) *a relation $r = \{h_1, \ldots, h_m\}$ over R.*

(Output:) *a minimal family of r.*

(Step 1:) *Find the equality set $E_r = \{E_{ij} : 1 \le i < j \le m\}$.*

(Step 2:) *Find the minimal generator N, where $N = \{A \in E_r : A \ne \cap\{B \in E_r : A \subset B\}\}$. Denote elements of N by $A_1, \ldots, A_t$.*

(Step 3: *For every $B \subseteq R$ if there is an A_i $(1 \le i \le t)$ such that $B \subseteq A_i$, then compute $C = \bigcap_{B \subseteq A_i} A_i$ and set $B \to C$. In the converse case set $B \to R$. Denote by T the set of all such functional dependencies*

(Step 4:) *Set $F = T - Q$, where $Q = \{X \to Y \in T : X \to Y$ is a redundant functional dependency $\}$.*

(Step 5:) *Put $M(Z_r) = \{(B, C) : B \to C \in F\}$.*

According to Theorem 1.2 and definition of $M(Z_r)$, Algorithm 2.10 finds a minimal family of r.

It can be seen that the time complexity of Algorithm 2.10 is exponential in the number of attributes.

Let $s =< R, F >$ be a relation scheme over R. We say that s is in Boyce-Codd normal form (BCNF) if $A \to \{a\} \notin F^+$ for $A^+ \ne R$, $a \notin A$.

If a relation scheme is changed to a relation we have the definition of BCNF for relation.

Proposition 2.11 *Given a BCNF relation r over R. The time complexity of finding a minimal family of r is exponential in the number of elements of R.*

Proof: From a given BCNF relation r we use Algorithm 2.10 to construct the minimal family of r. By definition of BCNF, we obtain

$M(Z_r) = \{(B, C) : B \to C \in F\} = \{(B, R) : B \in K_r\}$.

Let us take a partition $R = X_1 \cup \ldots \cup X_m \cup W$, where $|R| = n, m = [n/3]$, and $|X_i| = 3$ $(1 \le i \le m)$.

Set $M = (K^{-1})^{-1}$, i.e. K^{-1} is a set of minimal keys of M, we have

$M = \{C: |C| = n-3, C \cap X_i = \emptyset$ for some $i\}$ if $|W| = 0$,

$M = \{C: |C| = n-3, C \cap X_i = \emptyset$ for some i $(1 \le i \le m-1)$ or $|C| = n-4, C \cap (X_m \cup W) = \emptyset\}$ if $|W| = 1$,

$M = \{C: |C| = n-3, C \cap X_i = \emptyset$ for some i $(1 \le i \le m)$ or $|C| = n-2, C \cap W = \emptyset\}$ if $|W| = 2$.

It is clear that $3^{\lfloor n/4 \rfloor} < |K^{-1}|, |M| \leq m+1$.
Denote elements of M by $C_1, \ldots, C_t$.
Construct a relation $r = \{h_0, h_1, \ldots, h_t\}$ as follows:
For all $a \in R$ $h_0(a) = 0$, for $i = 1, \ldots, t$

$$h_i(a) = \begin{cases} 0 & \text{if } a \in C_i \\ i & \text{otherwise.} \end{cases}$$

Clearly, $|r| < |R|$ holds. According to Theorem 1.1 M is the set of antikeys of r and K^{-1} is the set of minimal keys of r. From definition of BCNF, we can see that $M(Z_r) = \{(B, R) : B \in K^{-1}\}$.

Thus, we can construct a relation r in which the number of rows of r is less than $|R|$, but the number of elements of $M(Z_r)$ is exponential in the number of attributes. □

Since the class of BCNF relations is a special subfamily of the family of relations over R, the next corollary is obvious.

Corollary 2.12 *The time complexity of finding a minimal family of a given relation r is exponential in the number of attributes.*

References

[1] Armstrong W.W., Dependency Structures of Database Relationships, *Information Processing 74, Holland Publ. Co.* (1974), 580-583.

[2] Beeri C., Bernstein P.A., Computational problems related to the design of normal form relational schemas, *ACM Trans. on Database Syst.* **4** (1979), 30-59.

[3] Beeri C., Dowd M., Fagin R., Staman R., On the Structure of Armstrong Relations for Functional Dependencies, *J. ACM.* **31** (1984), 30-46.

[4] Békéssy A., Demetrovics J., Contribution to the theory of data base relations, *Discrete Math.* **27** (1979), 1-10.

[5] Demetrovics J., Logical and structural investigation on relational datamodel *MTA-SZTAKI Tanulmányok, Budapest, Hungarian.* **114** (1980), 1-97.

[6] Demetrovics J., Thi V.D., Some results about functional dependencies, *Acta Cybernetica.* **8** (1988), 273-278.

[7] Demetrovics J., Thi V.D., Relations and minimal keys, *Acta Cybernetica.* **8** (1988), 279-285.

[8] Demetrovics J., Thi V.D., On Keys in the Relational Datamodel, *J. Inform. Process. Cybernet.* **24** (1988), 515-519.

[9] Demetrovics J., Thi V.D., On algorithms for generating Armstrong relations and inferring functional dependencies in the relational datamodel, Computers and Mathematics with Applications. To appear.

[10] Demetrovics J., Thi V.D., Armstrong Relation, Functional Dependencies and Strong Dependencies, *Comput. and AI.* To appear.

[11] Mannila H., Raiha K.J., Design by Example: An Application of Armstrong relations, *J. Comput. Syst. Scien.* **33** (1986), 126-141.

[12] Osborn S.L., Testing for existence of a covering Boyce-Codd normal form, *Infor. Proc. Lett.* **8** (1979), 11-14.

[13] Thi V.D., Investigation on Combinatorial Characterisations Related to Functional Dependency in the Relational Datamodel, *MTA-SZTAKI Tanulmányok, Budapest, Hungarian. (Ph.D. Dissertation).* **191** (1986), 1-157.

[14] Thi V.D., Minimal keys and Antikeys, *Acta Cybernetica.* **7** (1986), 361-371.

[15] Thi V.D., On the Antikeys in the Relational datamodel, *Alkalmazott Matematikai Lapok.* (in Hungarian) **12** (1986), 111-124.

[16] Thi V.D., Logical Dependencies and Irredundant Relations, *Computers and Artificial Intelligence.* **7** (1988), 165-184.

[17] Yu C.T., Johnson D.T., On the complexity of finding the set of candidate keys for a given set of functional dependencies, *Inform. Proc. Letters.* **5** (1976), 100-101.

Received July 29, 1993

Subscription information and mailing address for editorial correspondence:

Acta Cybernetica
Árpád tér 2.
Szeged
H-6720 Hungary

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Gécseg Ferenc
A kézirat a nyomdába érkezett: 1994. október
Terjedelem: 7,12 (B/5) ív